2010

# FPGA Implementation of Blind Source Separation using FastICA

Al-Laith Taha
*University of Windsor*

# FPGA Implementation of Blind Source Separation using FastICA

*By*

**AL-LAITH TAHA**

A Thesis

Submitted to the Faculty of Graduate Studies through the

Department of Electrical and Computer Engineering in Partial Fulfillment

of the Requirements for the Degree of Master of Applied Science at

The University of Windsor

Windsor, Ontario, Canada

2010

FPGA Implementation of Blind source Separation using FastICA

By

AL-LAITH TAHA

APPROVED BY:

W. Abdul-Kader

Department of Industrial and Manufacturing Systems Engineering

M. A. S. Khalid

Department of Electrical and Computer Engineering

E. Abdel-Raheem, Advisor

Department of Electrical and Computer Engineering

N. Kar, Chair of defense

Department of Electrical and Computer Engineering

University of Windsor

August 19, 2010

# *Author's Declaration of Originality*

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University of Institution.

# *Abstract*

Fast Independent Component Analysis (FastICA) is a statistical method used to separate signals from an unknown mixture without any prior knowledge about the signals. This method has been used in many applications like the separation of fetal and maternal Electrocardiogram (ECG) for pregnant women. This thesis presents an implementation of a fixed-point FastICA in field programmable gate array (FPGA). The proposed design can separate up to four signals using four sensors. QR decomposition is used to improve the speed of evaluation of the eigenvalues and eigenvectors of the covariance matrix. Moreover, a symmetric orthogonalization of the unit estimation algorithm is implemented using an iterative technique to speed up the search algorithm for higher order data input. The hardware is implemented using Xilinx virtex5-XC5VLX50t chip. The proposed design can process 128 samples for the four sensors in less than 63 ns when the design is simulated using 10 MHz clock.

# *Acknowledgments*

I would like to express my sincere appreciation to my supervisor, Dr. Esam Abdel-Raheem for his invaluable guidance and encouragement. He guided me throughout my thesis with great patience. I would also like to express my gratitude to the other members of my committee, Dr. Mohammed A. S. Khalid and Dr. W. Abdul-Kader, for their help and assistance.

I can't forget those days when I worked side by side with my fellow graduate students of the ECE department, Iman, Ishaq, Mohamed Islam. They give me a lot of help and encouragement during my study.

I would not forget my parents for their constant and unconditional support throughout my research.

Finally, my sincere appreciation to Canadian Microelectronics Corporation (CMC) for providing the computer and FPGA workstations for this research.

# *Table of Contents*

# *List of Figures*

# *List of Tables*

# *List of Abbreviations*

| Abbreviation | Definition |
|---|---|
| ADC | Analogue to digital converter |
| BSS | Blind source separation |
| DSP | Digital signal processing |
| ECG | Electrocardiogram |
| EEG | Electroencephalography |
| FECG | Fetal electrocardiogram |
| FPGA | Field program gate array |
| FASTICA | Fast independent component analysis |
| GMSC | Global maximum stopping criterion |
| ICA | Independent component analysis |
| IO | Input output |
| MC | Minor component |
| MECG | Fetal electrocardiogram |
| PCA | Principle component analysis |
| PC | Principle component |
| pdf | Probability density function |
| VHDL | Very high scale Hardware description language |

# Chapter 1

# *Introduction*

## 1.1 Background

If we consider the situation of attending a party, our ears capture numerous sounds: a friend's voice, the voices of others, background music, ringing telephones, and many others. If one concentrates, one can hear what a person is saying and you will filter any other sound. One can also change his/her focus of attention. For example, one may pay attention to your friend's speech first and shift focus to the music if it is playing a song you like. The ability to focus and recognize a specific source called the cocktail party effect [1, 2, 3]. If we were to record these sources by placing microphones in many places inside the room, the playback would be jumbled mix of sounds. One might be able to pick out a few words here and there, but there is no way one would be able to hear the conversation details. If there were as many microphones in the room as people, it is possible to extract and separate each individual conversation by using blind source separation algorithms [4]. This would allow us to hear everything in the room. In another words, Blind source separation (BSS) defined as the method that separate or estimate the original sources from an array of sensors or transducers without having any prior knowledge of the original sources [4]. BSS also is also a general class of signal processing methods that extract statistically independent source signals from linear mixtures with no or little information about the sources or the mixing conditions [5, 6]. In

instantaneous mixing, the mixtures are weighted sums of the individual source signals without dispersion or time delay, as shown in Fig. 1.1. Most of the mixtures in reality are added sources or sometimes called instantaneous mixtures.

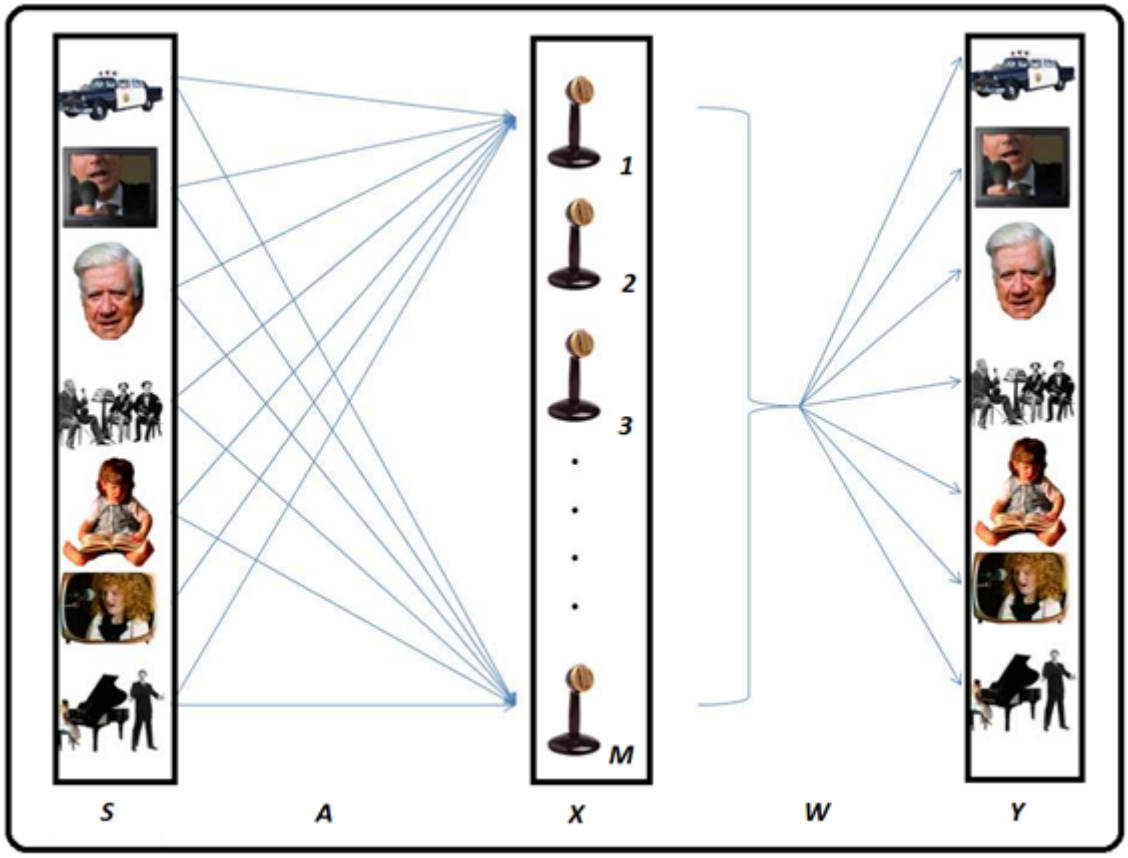


Figure 1.1: The instantaneous mixtures source separation example

In Figure 1.1, $\boldsymbol{S}$ refers to the original sources matrix $\boldsymbol{S}$, $\boldsymbol{A}$ is the mixing matrix and $\boldsymbol{X}$ is the observation matrix. The matrices $\boldsymbol{S}$ and $\boldsymbol{X}$ are both of size $M \times N$ matrices while the matrix $\boldsymbol{A}$ is of size $M \times M$. The values *M and N are* the number of sensors and the number of samples, respectively. The matrix $\boldsymbol{S}$ has the form:

$$\boldsymbol{S} = \begin{bmatrix} s_{11} & \cdots & s_{1N} \\ \vdots & \ddots & \vdots \\ s_{M1} & \cdots & s_{MN} \end{bmatrix} \tag{1.1}$$

The observation matrix $\boldsymbol{X}$ can be modeled as Follows:

$$\boldsymbol{X} = \boldsymbol{AS} \tag{1.2}$$

or

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x_1} \\ \boldsymbol{x_2} \\ . \\ \boldsymbol{x_M} \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1N} \\ \vdots & \ddots & \vdots \\ x_{M1} & \cdots & x_{MN} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1M} \\ \vdots & \ddots & \vdots \\ a_{M1} & \cdots & a_{MM} \end{bmatrix} \begin{bmatrix} s_{11} & \cdots & s_{1N} \\ \vdots & \ddots & \vdots \\ s_{M1} & \cdots & s_{MN} \end{bmatrix} \tag{1.3}$$

The coefficients of the mixing matrix $\boldsymbol{A}$ are unknown. The goal of the BSS algorithms is to find a demixing matrix $\boldsymbol{W}$ that has the following form:

$$\boldsymbol{Y} = \boldsymbol{W}\,\boldsymbol{X} \tag{1.4}$$

where $\boldsymbol{W}$ is an $M \times M$ matrix, $\boldsymbol{Y}$ and $\boldsymbol{X}$ are $M \times N$ matrices. The obtained estimated sources $\boldsymbol{Y}$ using BSS algorithms have certain unknown factors such as arbitrary scaling, permutation, and delay of estimated source signals. However, the most relevant information is contained in the waveforms of these signals, thus these unknown factors do not affect the separation if statistical methods like BSS are used [7]. Historically, principle component analysis (PCA) has been widely used for the same types of problems currently being investigated using BSS algorithms [3, 8]. The main difference between the two approaches is that BSS finds non-Gaussian and independent sources signals, whereas PCA finds sources, which are uncorrelated and have Gaussian distributions [3].

The performance of BSS algorithms such as FastICA is better in comparison with the PCA [5] because PCA does not work on super-Gaussian or sub-Gaussian distributions while FastICA does [6]. Since BSS does not require any prior knowledge about the mixed signals, BSS has attracted many areas of research. For example, in surveillance application where the goal is to find a specific voice among many [9, 10]. In addition, wireless communication has adopted BSS to suppress the co-channel interference in

multi-antenna system without requiring the receiving end to decode the signals [11]. The most useful application in utilizing the BSS techniques would be in the area of biomedical signal processing, where BSS is applied in Electrocardiography (ECG). Some of the applications involve the separation of the Mother ECG (MECG) from the fetal ECG (FECG) [12, 13]. In addition, some complex scenarios involve separating the MECG from a twin fetal [14]. Recently, Field programmable gate array (FPGA) technology has been the choice of implementation in the area of digital signal processing and neural networks. Kim *et al.* [15] implemented real-time blind source separation and adaptive noise cancellation for speech enhancement in FPGA. Du and Qi [16] implemented a parallel ICA on the Multi FPGA pilchard, a reconfigurable computing development environment dedicated for Sun Microsystems [18]. Celik *et al.* [17] implemented a mixed-signal real-time blind source separation that can only unmix two independent sources. The design in [17] is implemented using 0.5µm COMS technology. Kuo-kai *et al* [19] implemented full real-time FPGA based FastICA system and used extra circuitry to acquire the signals using two sensors. Two separate modules were used. The first module was used to acquire the mixed signals using an analogue to digital converter and filters. The second module converted the separated signals to analogue using an digital to analogue converter.

All the previously proposed FPGA implementation focused on implementing ICA algorithms with the assumption that only two signals are mixed and only two sensors are used to capture the mixed signals for separations. In addition, the previous work implemented the blind source separation algorithm using algebraic solutions. For higher dimension signals, implementing the BSS algorithm algebraically is too complex. In

addition, it can result in a very large propagation delay and, in turn, affects the overall system performance.

## 1.2 FPGA background

FPGA is a large-scale integrated circuit that can be programmed after it is manufactured rather than being limited to a predetermined unchangeable hardware function. FPGA technology is widely used in digital signal processing [20]. It combines the speed of dedicated blocks, application-optimized hardware and reprogrammability of microprocessors, which makes it suitable for high speed implementation of blind source separation. FPGA has been the choice of implementation of most of digital signal processing algorithms. DSP algorithms can be designed, tested and implemented on an FPGA chip without any fabrication delays. FPGAs consist of the following elements:

1. Programmable logic cells which provide the functional elements for construction of the user's input.
2. Input output (IO) Blocks which provide the interface between the logic cells and the output pins.
3. Programmable interconnects which provide the routing paths to connect the input and output of logic cells and the IO pins.

Modern FPGAs provide high-level arithmetic and control structures, such as multipliers, counters, multiply accumulate units, memory resources and processor cores. These resources provide high performance, low power consumption and are highly suitable for DSP applications [21]. The behavior of an FPGA can be defined by using a hardware description language (HDL) such as VHDL or Verilog or by arranging blocks of existing functions using a schematic-oriented design tool. The design is compiled and

synthesized using proprietary FPGA place-and-route tools. The compilation and synthesis process generates a bit file that can be downloaded on the FPGA [21, 22].

Although FPGAs are similar to custom Application-specific integrated circuit (ASIC) design, FPGAs can implement and test proposed designs instead of sending them to the manufacturer and wait for the chip to be tested afterwards.

## 1.3 Thesis objective

The thesis focuses on the following areas:

1. Investigating different types of independent component analysis (ICA) algorithms for non-Gaussian signals and compare the results with principle component analysis (PCA).
2. Developing an efficient numerical solution instead of the algebraic solution for FastICA.
3. Investigating the development of FastICA algorithm using different types of orthogonalization techniques.
4. Implementing FastICA algorithm in XILINX virtex5-XC5VLX50t FPGA.

The main challenge in this thesis is implementing the ICA algorithm for higher order data inputs. The complexity of the circuit grows exponentially depending on the size of the demixing matrix $\boldsymbol{W}$. In addition, FPGA is used to implement the algorithm.

## 1.4 Thesis organization

This thesis is organized as follows: Chapter 2 provides the mathematical analysis of the ICA algorithm, which is FastICA. Chapter 3 describes the proposed FastICA model using QR method and the symmetrical orthogonalization as well as the hardware implementation of the proposed algorithm using FPGA technology, Chapter 4 provides

simulations of the proposed hardware and presents two experiments while Chapter 5 provides concluding remarks and suggestions for future work to enhance the design.

# Chapter 2

# *Independent Component Analysis*

## 2.1 Introduction

This Chapter provides the mathematical theory behind ICA method. ICA is one of a family of techniques, including PCA and blind deconvolution, for solving the BSS problems. ICA is a method for finding underlying factors in a multidimensional data. This Chapter also explains the PCA method and its limitations. Finally, a special ICA algorithm called FastICA is presented and compared with the PCA method.

## 2.2 General statistical settings

The main goal of any statistical model is to find a suitable representation of the parameters of a multivariable system that render the essential structure governing the variables more visible. This usually presents computational and representational obstacles that must be tackled.

To illustrate the above, a linear system is shown in Fig. 2.1. In the system, every input vector in $\boldsymbol{X}$ contains a linear combination of observed sensor samples of size $N$ as in Equation (1.4) given in Chapter 1. In the figure, $\boldsymbol{W}$ is the demixing matrix of size $M \times M$. The aim, as explained in Chapter 1, is to determine the output matrix $\boldsymbol{Y}$. However, the system contains unknown sources with no prior knowledge of the noise type and its contribution to the system.

Moreover, the difficulty of this model lies in the fact that it requires determining the elements of the $\boldsymbol{W}$ matrix using linear algebra, which finds a simple solution to $\boldsymbol{W}$ only by assuming the signals independence [23].

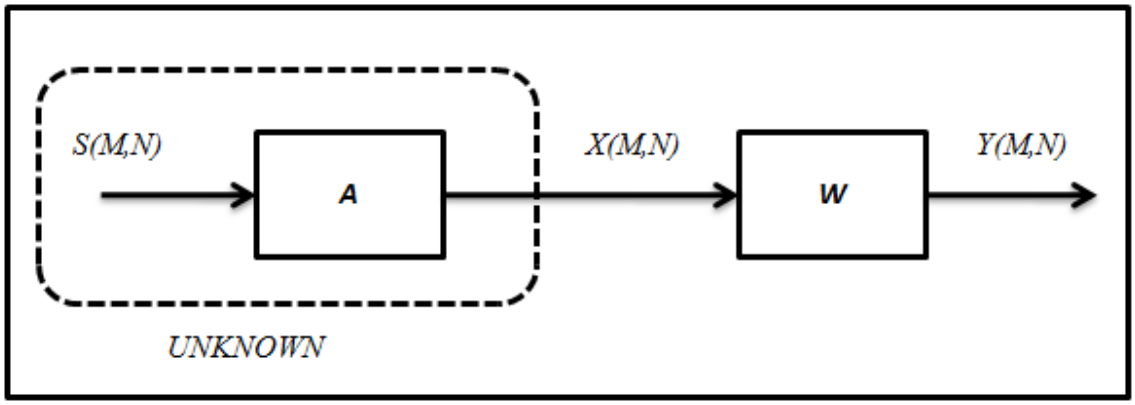


Figure 2.1: Linear instantaneous BSS problem

Fig. 2.2 shows four source signals that result in the mixture shown in Fig. 2.3 when mixed together. The problem is that the original signals information is not usually available. In fact, it is nearly impossible to know what these signals might contain when they are mixed. But with the aid of BSS techniques, it is possible to extract or at least estimate the hidden signals. For example, considering matrices $\boldsymbol{W}$ and $\boldsymbol{X}$ of sizes $4 \times 4$ and $4 \times N$, respectively, using statistical independence only, the original signals in Fig. 2.2 can be estimated by multiplying $\boldsymbol{W}$ by $\boldsymbol{X}$ as follows:

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{y_1} \\ \boldsymbol{y_2} \\ \boldsymbol{y_3} \\ \boldsymbol{y_4} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix} \begin{bmatrix} x_{11}\, x_{12}\, x_{13} \cdots x_{1N} \\ x_{21}\, x_{22}\, x_{23} \ldots x_{2N} \\ x_{31}\, x_{32}\, x_{33} \cdots x_{3N} \\ x_{41}\, x_{42}\, x_{43} \ldots x_{4N} \end{bmatrix} \tag{2.1}$$

As a result, $\boldsymbol{Y}$ contains four vectors that are the separated signals which are the result of estimation using only the information of the signals. Fig. 2.3 shows clearly four

distinct signals that are not dependent of the other. The separated signals are easily distinguished as square, sin, sawtooth, and random noise waves.

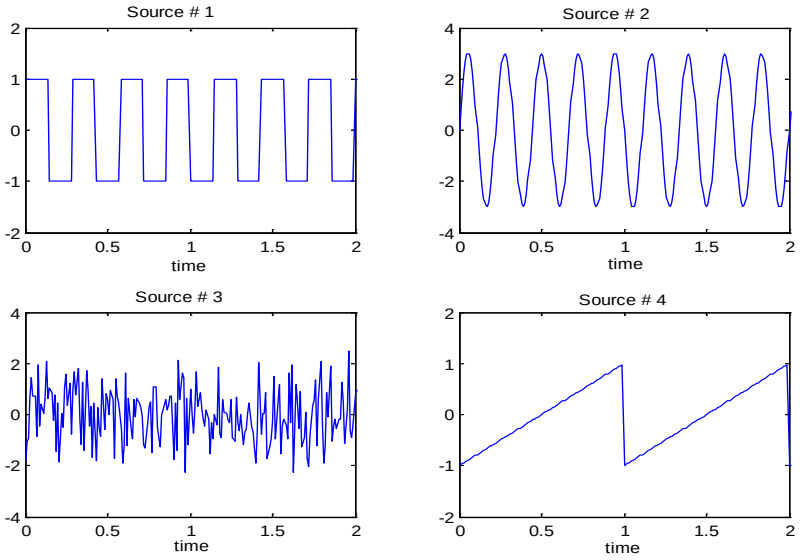


Figure 2.2: Sources before mixing

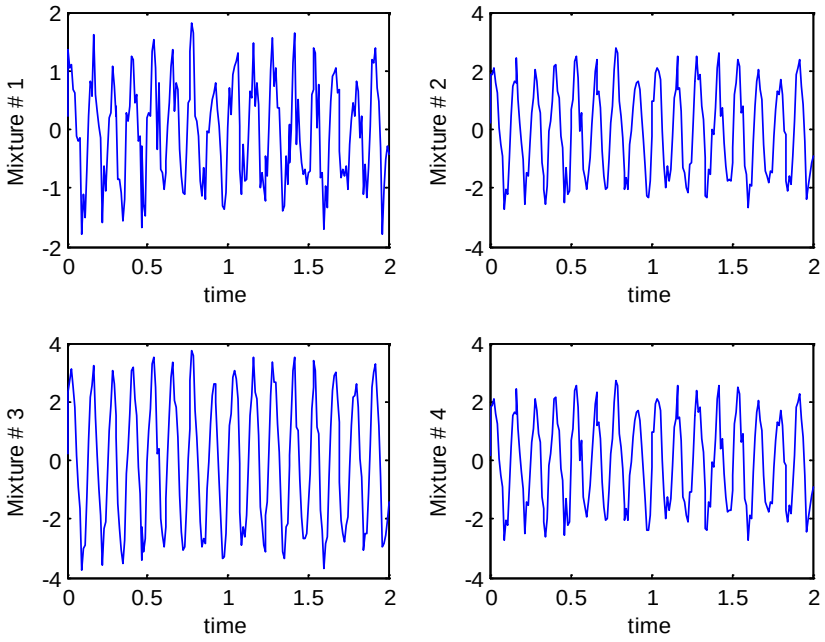


Figure 2.3: Mixed Signals that contain some underlying hidden factors

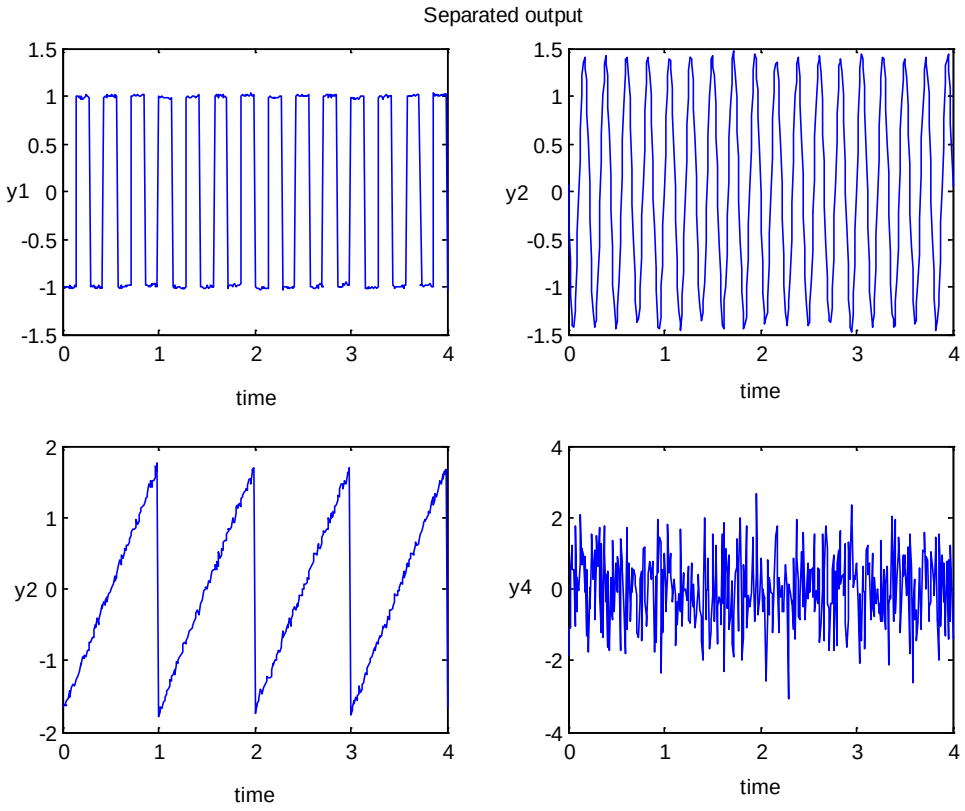


Figure 2.4: Separated Signals

## 2.3 Principle component analysis (PCA)

PCA has been widely used in pattern recognition and signal processing [24]. The algorithm decomposes a set of mixed signals into a set of uncorrelated signals [7]. Given a set of multivariate measurements, the purpose of the PCA is to find a smaller set of variables with less redundancy that would result in a good representation of the data. PCA can classify signals based on the mixture statistical information (variances.). Each principle component (PC) represents a cluster of information in the mixture. The PC that has the highest variance is referred as the major component while those components with the smallest variances called the minor components [25]. If the PCs contain high statistical information (high variances), it means that those PCs contains real signals and

if the PCs contain very low variances, it is an indication that the mixture contains unwanted signals like noise or interference [25].

### 2.3.1 PCA algorithm

PCA transforms a process such that the data are represented along a new set of orthogonal dimensions with a diagonal covariance matrix [24]. In addition, the PC coefficient with the largest variance is the first principle component; the PC coefficient with the second largest variance is the second most important and so on. The PCA algorithm consists of the following steps [24]:

1. Centering: Centering is used as a preprocessing in PCA. Centering is the process of calculating the mean of the observation matrix $\boldsymbol{X}$ and subtracting it from the source. It can be defined as:

$$\boldsymbol{X_{cen}} = \boldsymbol{X} - \boldsymbol{uh} \tag{2.2}$$

where $\boldsymbol{X_{cen}}$ is the centered observation matrix and has the same dimension as $\boldsymbol{X}$. The $1 \times N$ $\boldsymbol{h}$ vector of all 1s, i.e.,

$$\boldsymbol{h}[n] = 1 \qquad \text{for } n = 1, \dots, N \tag{2.3}$$

Moreover, $\boldsymbol{u}$ is an $M \times 1$ vector which is the empirical mean of $\boldsymbol{X}$ and can be calculated as:

$$u[m] = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{X}(m,n) \quad \text{for } m = 1, \dots, M \tag{2.4}$$

2. Calculating Covariance matrix and its eigenvalues and eigenvectors: The PCA algorithm is based on calculating the eigenvalues and eigenvectors of the $M \times M$ covariance matrix $\boldsymbol{C_x}$ which is defined as [25].

$$\boldsymbol{C_x} = E[\boldsymbol{X}\boldsymbol{X}^{\boldsymbol{T}}] \tag{2.5}$$

where E[·] is the expectation operation. The covariance matrix is used to compute the eigenvectors. Each eigenvector corresponds to a specific eigenvalue. Since the covariance matrix is real and symmetric, the eigenvectors are real and orthonormal [26]. Traditionally, the eigenvalues of a matrix is calculated algebraically using the following steps [16], [19]:

a) Find the characteristic equation of $\boldsymbol{C_x}$ by setting $\det(\boldsymbol{C_x} - \lambda_m \boldsymbol{I}) = 0$ of the covariance matrix where $\boldsymbol{I}$ is an identity matrix that has the same dimensions as $\boldsymbol{C_x}$ and $\lambda_m$ are the eigenvalues to be found.

b) Find the roots of the characteristic equation which are the eigenvalues of $\boldsymbol{C}_{\mathrm{x}}$.

The complexity of finding the roots of the characteristic equation increases when the order of $\boldsymbol{C_x}$ increase. Most of the previous separation models using PCA approach use only $2 \times 2$ matrices [27, 28], which result in second order polynomials [25]. For higher order matrices, an iterative numerical solution is used [29]. The most common numerical methods used to find the eigenvalues and eigenvectors for higher order matrices are the upper triangular matrix, the power method, the orthogonal iteration, the QR decomposition and the singular value decomposition [29, 30].

4. Whitening: which is the last stage in the PCA technique, it forces the sources in the mixture to be uncorrelated but with a unit variance [25]. The whitening matrix $\boldsymbol{V}$ can be expressed in terms of the eigenvalues and eigenvectors of $\boldsymbol{C}_{\mathrm{x}}$ as follows [25]:

$$\boldsymbol{V} = \boldsymbol{D}^{-1/2}\boldsymbol{E}^{T} \tag{2.6}$$

where $\boldsymbol{E}$ is an $M \times M$ matrix containing all the eigenvectors of $\boldsymbol{C_x}$, while $\boldsymbol{D}$ is an $M \times M$ diagonal matrix with the values in the diagonal comprising the eigenvalues of $\boldsymbol{C_x}$. The

matrix $\boldsymbol{V}$ is also called the square root of the covariance matrix, i.e. $\boldsymbol{C_x}^{-1/2}$ [34].

The last step in the PCA is to find the uncorrelated signals $\boldsymbol{Z}$ using the following equation:

$$\boldsymbol{Z} = \boldsymbol{V}\boldsymbol{X_{cen}} \tag{2.7}$$

where $\boldsymbol{Z}$ and $\boldsymbol{X_{cen}}$ are $M \times N$matrices. In general, $\boldsymbol{V}$ solves half of the ICA problem which means forcing the signals to be uncorrelated and transforms the signals orthogonally [31]. In most applications, this is not sufficient to ensure that the signals are independent, which is why whitening solves only half of the ICA problem. However, the whitening step reduces the computations of separation by half [34]. More specifically, the orthogonal nature of $\boldsymbol{V}$ reduces the problem from finding $k^2$ parameters which are the elements of $\boldsymbol{V}$ to finding only $k\ (k-1)/2$ parameters [32].

### 2.3.2 Whitening limitations

Assume that the data in the ICA model is whitened using Equation (2.7). The whitening matrix transforms the mixing matrix $\boldsymbol{A}$ in equation (1.2) into a new mixing matrix called $\boldsymbol{\grave{A}} = \boldsymbol{VA}$ so that the new ICA model is written as follows:

$$\boldsymbol{Z} = \boldsymbol{VX} = \boldsymbol{VAS} \tag{2.8}$$

Unfortunately, whitening cannot solve the ICA problem, since whiteness or uncorrelatedness does not imply independence [28]. Uncorrelatedness is weaker than independence, and is not by itself sufficient to estimate any ICA model [32].

On the other hand, whitening is useful as a preprocessing step in ICA. The usefulness of whitening resides in the fact that the new mixing matrix $\boldsymbol{\grave{A}}$ is orthogonal [28, 33]. This means that we can restrict the search in the mixing matrix to the space of orthogonal matrices. That means instead of estimating $k^2$ parameters that are the elements of the

original matrix $\boldsymbol{A}$ [7], we only need to estimate an orthogonal mixing matrix $\boldsymbol{\grave{A}}$. Thus, it could be said that whitening solves half of the ICA problem because whitening is a very simple and standard procedure, much simpler than any ICA algorithms. The remaining half of ICA can be estimated by some other methods like FastICA [34], which is the focus of this thesis. However, PCA can be used as a preprocessing step before the ICA algorithms [31].

## 2.4 Higher order statistics

Most of the standard methods in signal processing systems utilize system's statistical information in linear discrete-time system. Although their theory is well defined and developed [35-38], these methods are utilizing the second order statistics and are driven by the assumptions of the source signals being stationary and are jointly governed by a Gaussian linear underlying system. Recently, an interest in the higher order statistics has began to grow in the signal processing area. At the same time, neural network has grown popular with the development of several new, efficient learning algorithms [32, 23, 39]. Neural networks consist of computational blocks called neurons. The output of the neurons depends nonlinearly on the input [40]. An example of the nonlinearity is the hyperbolic tangent $\tanh(\boldsymbol{U})$, the matrix $\boldsymbol{U}$ is of size $M \times N$ which is the inner product $\boldsymbol{U = WX}$**.** It introduces nonlinearity to the process [40]. ICA requires the use of higher order statistics via nonlinearities [33, 40]. In the following, the concept of kurtosis and its role in the higher orders statistics [33].

### 2.4.1 Central moments and kurtosis

The mean of the data vector $\boldsymbol{z}$ is defined as:

$$u = E\{\boldsymbol{z}\} \tag{2.9}$$

where $\boldsymbol{z}$ is a vector in the whitened data matrix $\boldsymbol{Z}$. In addition, the $j^{th}$ central moment is defined as:

$$u_j = E\{(\boldsymbol{z} - u)^j\} \tag{2.10}$$

The second central moment is the standard deviation of the whitened data $\boldsymbol{Z}$ denoted as $\sigma^2$. The third central moment is called skewness and it will be used in this thesis. However, the fourth moment has been intensively used in the area of blind source separation [32, 35, 39]. Moments that are higher than 4th order are rarely used in practice [32] and will not be discussed in the thesis.

The fourth moment on the other hand, is simple and effective in some BSS algorithm like FastICA. The fourth central moment is [32]:

$$u_4 = E[(\boldsymbol{z} - u)^4] \tag{2.11}$$

The fourth central moment is also called the Kurtosis and can be rewritten as [33].

$$Kurt(\boldsymbol{z}) = E[\boldsymbol{z}^4 - 3[E[\boldsymbol{z}^2]]^2 \tag{2.12}$$

We can also rewrite it in the following form [31]:

$$Kurt(\boldsymbol{z}) = \frac{E[\boldsymbol{z}^4]}{E[\boldsymbol{z}^2]^2} - 3 \tag{2.13}$$

For whitened data $E[\boldsymbol{z}^2] = 1$, the Kurtosis is reduced to the following [38]:

$$Kurt(\boldsymbol{z}) = E[\boldsymbol{z}^4] - 3 \tag{2.14}$$

This implies that for the whitened data, the fourth order moment can be used instead of the Kurtosis to represent the fourth order central moment of $\boldsymbol{Z}$. The most important property of the kurtosis is that it has the ability to detect non-Gaussian signals. If the kurtosis is zero, it implies that the distribution is Gaussian. If the Kurtosis is negative, the distribution is sub Gaussian. If the kurtosis is positive, the distribution is super-Gaussian.

However, the absolute value of the Kurtosis is used for simplicity since it is only needed to know if the signal is non-Gaussian or not [33, 38, 39, 40].

### 2.4.2 Fixed-point FastICA algorithm using kurtosis

In the previous section, the Kurtosis has been introduced as a measure of non-Gaussianity [32]. The advantage of such technique can be adapted by neural networks [36]. However, the convergence is slow and the choice of the input sequence has to be chosen carefully [40]. A bad choice of the input sequence would lead to divergence. Alternatively, the fixed-point iterative algorithm that has been developed by Hyvarinen and Oja is used [39]. To achieve a more efficient fixed-point iteration, the gradient must point to the direction of the weight vector $\boldsymbol{w_m} = [w_1, w_2, \dots, w_M]^T$. The gradient must equal to $\boldsymbol{w_m}$ multiplied by some value. As a result, the weight vector $\boldsymbol{w_m}$ can be written as [39]:

$$\boldsymbol{w_m} = [E\{\boldsymbol{Z}(\boldsymbol{w_m^T Z})^3\} - 3\|\boldsymbol{w_m}\|^2 \boldsymbol{w_m}] \tag{2.15}$$

Equation (2.15) is further simplified as a fixed-point iteration algorithm by computing the right hand side and assign the new value to $\boldsymbol{w_m}$. Thus Equation (2.15) can be rewritten as follows [39]:

$$\boldsymbol{w_m} \leftarrow E\{\boldsymbol{Z}(\boldsymbol{w_m^T Z})^3\} - 3\boldsymbol{w_m} \tag{2.16}$$

$$\boldsymbol{w_m} \leftarrow \boldsymbol{w}_{\mathrm{m}} / \|\boldsymbol{w}_{\mathrm{m}}\| \tag{2.17}$$

The weight vector $\boldsymbol{w_m}$ is divided by its norm using Equation (2.17) after every iteration in the FastICA, is a necessary normalization step to keep the variance of the term $\boldsymbol{w}_m^T \boldsymbol{Z}$ constant [33]. If the PCA is considered as a preprocessing stage prior to the FastICA, the FastICA algorithm would have the following steps:

1. Center the input $\boldsymbol{X}$.
2. Whiten the $\boldsymbol{X}_{\mathrm{cen}}$ matrix to give $\boldsymbol{Z}$.

3. Choose $M$ the number of independent components to be estimated.
4. Initialize the first vector $\boldsymbol{w}$ to any random numbers.
5. Run the FastICA algorithm on $\boldsymbol{w}$.
6. Normalize $\boldsymbol{w}$ by dividing it by its norm.
7. If $\boldsymbol{w}$ has not converged, go back to step 3.

where $\boldsymbol{w}$ is a vector in $\boldsymbol{W} = [\boldsymbol{w_1}, \boldsymbol{w_2}, \boldsymbol{w_3}, \dots, \boldsymbol{w_M}]^{\boldsymbol{T}}$ . However, it can be noticed that the algorithm searches for a single weight vector in $\boldsymbol{W}$ which means only one signal can be estimated. That is why this method is called one-unit FastICA [32]. To estimate the other weight vectors $\boldsymbol{w_M}$, an orthogonalization step is needed after the search has converged to the first weight vector $\boldsymbol{w_1}$ [40]. Otherwise, the search might converge to the same maxima [34] if other initial values were to be applied in step 2. Actually, this iterative technique has a very fast convergence and reliable [30]. The algorithm has two main superior advantages over the normal gradient-based algorithms. Firstly, the convergence of this algorithm is cubic. It implies that the convergence is rapid [31]. Secondly, this algorithm has no learning rate or other adjustable parameters [32].

## 2.5 FastICA using orthogonalization techniques

So far, the search algorithm that has been discussed finds one component in the mixture. In most of time, $\boldsymbol{X}$ has more than one component, that is why it is necessary to account for the other components in the weight matrix $\boldsymbol{W}$. Also, the search algorithms don't usually converge to orthogonal results as in theory [32], which is why orthogonalization must be applied at every step in FastICA [41]. The key concept of orthogonalization is that the weight matrix $\boldsymbol{W}$ corresponds to the different components in the projection

subspace. The most common techniques are the Gram-Schmidt or sometimes called deflationary orthogonalization method and the symmetric orthogonalization [32].

### 2.5.1 FastICA using deflationary orthogonalization

Deflationary orthogonalization is simple and the oldest technique in orthogonalization [41]. It estimates the independent components one by one using Gram-Schmidt method. Followed by running the one-unit FastICA for $\boldsymbol{w}_m$ where $m$ is the number of independent components $m = 1,..,M$ . After every iteration, projections $(\boldsymbol{w}_{m+1}^T \boldsymbol{w}_i)\boldsymbol{w}_i$ where $i = 1,\dots,M$ of the previously estimated $m$ vectors is subtracted from $\boldsymbol{w}_{m+1}$ [31]. After the first successful calculation, the values of the first weight vector $\boldsymbol{w}_1$ are obtained. Similarly, after the $m^{\text{th}}$ iteration , the values of the corresponding vector $\boldsymbol{w}_m$ are obtained. The resulting values of all the vectors obtained from the iterations are placed in the final unmixing matrix $\boldsymbol{W}$ of size $M \times M$.

It is worth noting that for simplicity purposes, an intermediate matrix $\boldsymbol{B}$ is used by the algorithm to hold the values of the generated $\boldsymbol{w}_m$ as they are obtained in the corresponding iteration. When the final iteration is complete, the final separation matrix $\boldsymbol{W}$ is equivalent to $\boldsymbol{B}$.

The final output matrix $\boldsymbol{Y}$ is then obtained by multiplying $\boldsymbol{W}$ by the whitened data matrix $\boldsymbol{Z}$ [32, 34, 42, 43].

The deflationary orthogonalization can be added to the one-unit FastICA so that the algorithm can separate $M$ independent components using the following steps [30]:

1. Center the input $\boldsymbol{X}$ so that it has zero mean.
2. Whiten the $\boldsymbol{X}_{cen}$ matrix to give $\boldsymbol{Z}$.
3. Choose $M$, the number of independent components to be estimated. (set $m = 1$).

4. Initialize the vector $\boldsymbol{w}_m$ to any random numbers.

5. Initiate the FastICA algorithm on $\boldsymbol{w}_m$.

6. If $\boldsymbol{w}_m$ has not converged, go back to step 3.

7. Start the deflationary orthogonalization using the following Equation:

$$\boldsymbol{w_m} \leftarrow \boldsymbol{w}_m - \sum_{j=1}^{m-1}\left(\boldsymbol{w}_m^T \boldsymbol{w}_j\right)\boldsymbol{w}_j \tag{2.18}$$

8. Normalize $\boldsymbol{w}_m$ by dividing it by its norm.

$$\boldsymbol{w}_m = \boldsymbol{w}_m / \|\boldsymbol{w}_m\| \tag{2.19}$$

9. Set $m \leftarrow m+1$. If m is not greater than the desired number of IC, go back to step 2.

The norm in step 7 is the second norm [32]. The Deflationary orthogonalization is shown in Fig. 2.5. It is clear that the process is serial, which indicates that the weight vectors $\boldsymbol{w}_M$ are calculated sequentially.

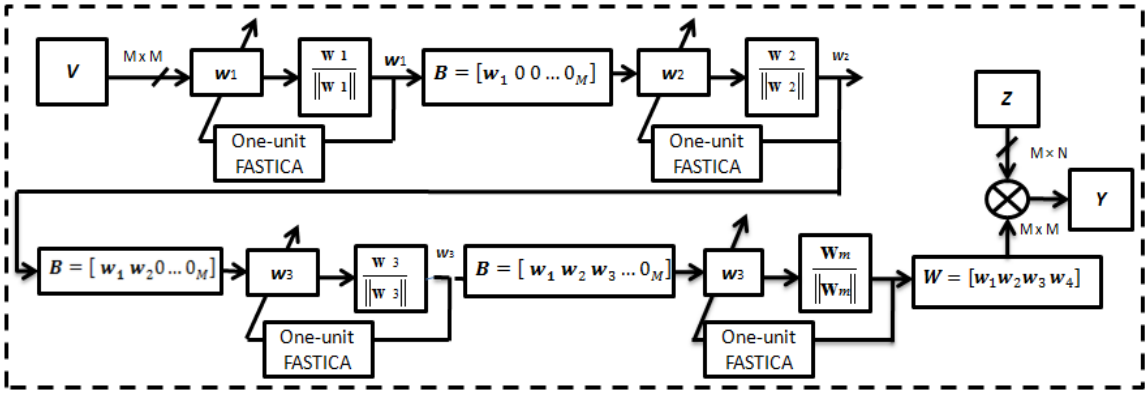


Figure 2.5: Deflationary orthogonalization block diagram

### 2.5.2 FastICA using symmetric orthogonalization

In some cases, sequential orthogonalization like the deflationary approach is not suitable for implementation [28]. Symmetric orthogonalization on the other hand finds the orthogonal vectors $\boldsymbol{w}_m$ that are the vectors of $\boldsymbol{W}$ in parallel. Symmetric orthogonalization is performed by first initiating the iterative step of the one-unit algorithm on $\boldsymbol{W}$, followed

by orthogonalizing $\boldsymbol{W}$ using symmetrical method. The symmetrical orthogonalization is performed using the following equation [30, 32, 36]:

$$\boldsymbol{W} \leftarrow (\boldsymbol{W}\boldsymbol{W}^T)^{-1/2}\boldsymbol{W} \tag{2.20}$$

In other words the FastICA steps using the symmetrical orthogonalization can be described as:

1. Center the input $\boldsymbol{X}$ so that it has zero mean.
2. Whiten the $\boldsymbol{X}_{\text{cen}}$ matrix to give $\boldsymbol{Z}$.
3. Choose $M$, the number of Independent components to be estimated. (Set $m = 1$).
4. Initialize the vector $\boldsymbol{w}_m \;\; m = 1, \dots, M$ to any random numbers.
5. Initiate the FastICA algorithm on every $\boldsymbol{w}_m$ in parallel.
6. Perform a symmetric orthogonalization of the matrix $\boldsymbol{W} = [\boldsymbol{w_1}, \dots, \boldsymbol{w_m}]^T$ using Equation (2.20).
7. Normalize $\boldsymbol{W}$ by dividing it by its norm.
8. If $\boldsymbol{W}$ has not converged, go back to step 3.

The inverse square root $(\boldsymbol{W}\boldsymbol{W}^{\boldsymbol{T}})^{-1/2}$ is obtained from the eigenvalue decomposition of $(\boldsymbol{W}\boldsymbol{W}^{\boldsymbol{T}}) = \boldsymbol{E}\, diag(d_1, \dots, d_m)\, \boldsymbol{E}^T$ [29], where $\boldsymbol{E}$ is an $M \times M$ matrix that contains the eigenvectors of $(\boldsymbol{W}\boldsymbol{W}^{\boldsymbol{T}})$ and $\left({d_1}^{-1/2}, \dots, {d_m}^{-1/2}\right)$ are the eigenvalues of $(\boldsymbol{W}\boldsymbol{W}^{\boldsymbol{T}})$. The eigenvalue decomposition can be further expanded as [38]:

$$(\boldsymbol{W}\boldsymbol{W}^{\boldsymbol{T}})^{-1/2} = \boldsymbol{E}\, diag\left({d_1}^{-1/2}, \dots, {d_m}^{-1/2}\right)\boldsymbol{E}^T \tag{2.21}$$

Fig. 2.6 shows FastICA algorithm using symmetrical orthogonalization. The algorithm starts by initializing $\boldsymbol{W}_{initial}$ to some random values. The orthogonalization is performed after every iteration in the FastICA. The FastICA algorithm is monitored by two parameters $\varepsilon$ and the maximum number of iterations [32].

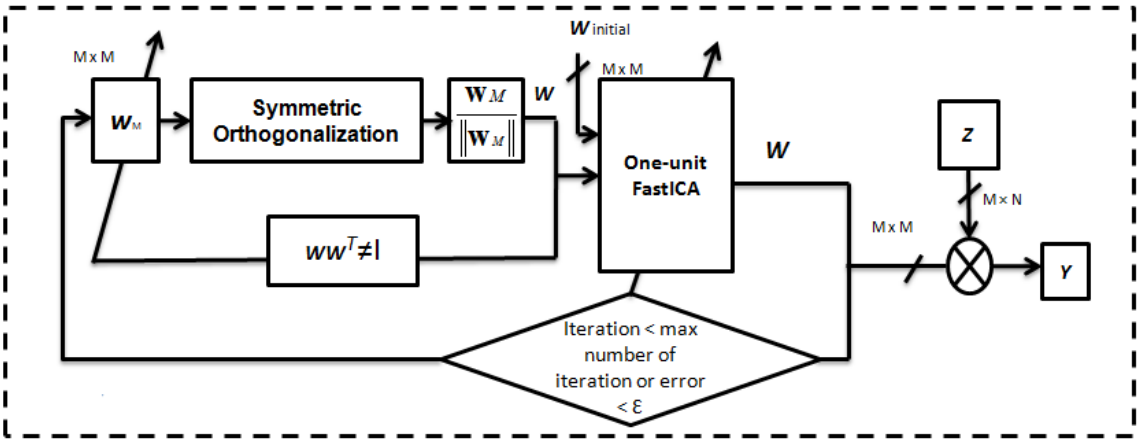


Figure 2.6: FastICA algorithm using Symmetric Orthogonalization

Notwithstanding, this method has some practical limitation due to the complexity of the matrix inversion calculation [30], especially when the order of $\boldsymbol{W}$ is high. An alternative iterative approach reported in [30, 32, 38] is used to solve this issue and is described by:

1. $\boldsymbol{W} = \boldsymbol{W}/\|\boldsymbol{W}\|$ (2.22)
2. $\boldsymbol{W} = \frac{3}{2}\boldsymbol{W} - \frac{1}{2}\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{W}$ (2.23)
3. If $\boldsymbol{W}\boldsymbol{W}^{\boldsymbol{T}}$ is not close enough to the identity matrix go back step 2.

The technique starts with a non-orthogonal matrix $\boldsymbol{W}$. The iterations continue until $\boldsymbol{W}\boldsymbol{W}^{\mathrm{T}} \sim \boldsymbol{I}$ is achieved. The convergence of the method is proven in Appendix **A**.

The advantage of this technique relies on the fact that matrix inversion is computationally intensive and calculating Equation (2.20) in every loop in the FastICA renders the hardware slow and inefficient [39]. Instead, Equations (2.22) and (2.23) are used to replace Equation (2.20). The norm in Equation (2.22) can be any norm, but for simplicity the second norm is used which is the maximum summation of the largest absolute value of any row or column in the weight matrix $\boldsymbol{W}$ [30, 41, 42].

## 2.6 Summary

In this chapter, the PCA and ICA models have been explained. Independent component analysis (ICA) is a method for finding underlying factors or components from multivariate (multidimensional) statistical data. What distinguishes ICA from other methods is that it looks for components that are both statistically independent and non-Gaussian where the PCA just decorelates the signals based on their variances. This chapter also explained the FastICA algorithm and how to use orthogonalization techniques to find all components of $\boldsymbol{W}$. In addition, deflationary and symmetrical orthogonalizations methods were discussed.

# Chapter 3

# *Proposed Architecture and FPGA Implementation*

## 3.1 Introduction

Having presented the PCA and FastICA algorithms in Chapter 2, I now detail the implementation process of the FastICA using PCA as a preprocessing stage. In Section 3.2, the proposed model is explained, after which the realization of the eigenvalues and eigenvectors is given in Section 3.3. The FastICA orthogonalization process is explained in Section 3.4. Section 3.5 gives the details of the hardware implementation while Section 3.6 presents the final implementation of the algorithm. The chapter ends with a summary in section 3.7.

## 3.2 Proposed model

The proposed BSS model accepts up to four input sensors. This assumption is interrupted as the model can separate up to four mixed signals in the mixture. The sources in the mixture are assumed to be independent and non-Gaussian. Let $\boldsymbol{Y}$ describes the separated signals in Equation (2.1) with the model being expanded to account for 4 signals. The number of samples is set to $N = 128$. The components of $\boldsymbol{Y}$ are estimated by multiplying the $4 \times 4$ unmixing matrix $\boldsymbol{W}$ by the $4 \times 128$ input matrix $\boldsymbol{X}$. It is noted that the input signals are also assumed to be independent and non-Gaussian.

$$\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{X} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & x_{13} \cdots & x_{1,128} \\ x_{21} & x_{22} & x_{23} \ldots & x_{2,128} \\ x_{31} & x_{32} & x_{33} \cdots & x_{3,128} \\ x_{41} & x_{42} & x_{43} \ldots & x_{4,128} \end{bmatrix} \tag{3.1}$$

Equation (3.1) shows the mathematical model used in the separation of the mixed signals. The goal of the ICA algorithm is to estimate the unmixing matrix $\boldsymbol{W}$. To do so, the PCA algorithm is first applied to force the signals to be uncorrelated and then the FastICA algorithm is applied. However, calculating the whitening matrix is not straightforward since the $\boldsymbol{C_x}$ have the same dimension as $\boldsymbol{W}$, i.e.

$$\boldsymbol{C_x} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix} \tag{3.2}$$

Finding the eigenvalues and eigenvectors algebraically for large covariance matrices such as Equation (3.2) is not computationally efficient. Instead, numerical solution is used in implementation [28]. However, only few iterative techniques can converge to find all the eigenvalues of the required matrices. For example, the power method finds only the dominant eigenvalue. Moreover, the convergence of the power method is the eigenvalues convergence is too slow and is not suitable for implementation [28]. The only simple and robust numerical solution that can find all eigenvalues and eigenvectors is the QR decomposition method [43]. However, the method works only on symmetric and positive definite matrix, fortunately, the $\boldsymbol{C_x}$ have these two properties [44, 45].

Since the covariance matrix is symmetrical, the only elements in the covariance matrix that are not repeated are the diagonal elements. Hence, the covariance matrix can be put in the form:

$$\boldsymbol{C_x} = \begin{bmatrix} c_{11} & c_{01} & c_{02} & c_{03} \\ c_{01} & c_{11} & c_{12} & c_{13} \\ c_{02} & c_{12} & c_{33} & c_{23} \\ c_{03} & c_{13} & c_{23} & c_{33} \end{bmatrix} \tag{3.3}$$

### 3.3 Realization of eigenvalues and eigenvectors

In this thesis, the QR decomposition is used to find the eigenvalues and eigenvectors numerically instead of finding them algebraically. Figure 3.1 shows the flow chart that describes the QR decomposition method [26].

The value $K$ is the number of the maximum iterations. There is no specific value for $K$, however, QR decomposition can achieve good result if the value of $K$ is more than 10 [26]. Nevertheless, in this work, it is decided to set $K = 20$ so that the result of the QR decomposition is approximately close to 3-significant figures. The matrices $\boldsymbol{R}$ and $\boldsymbol{Q}$ that result from the method are both of size $M \times M$. High-precision approximation is required because the hardware implementation is carried out using fixed-point number system and the error that builds in the calculation may affect the result of the QR decomposition method.

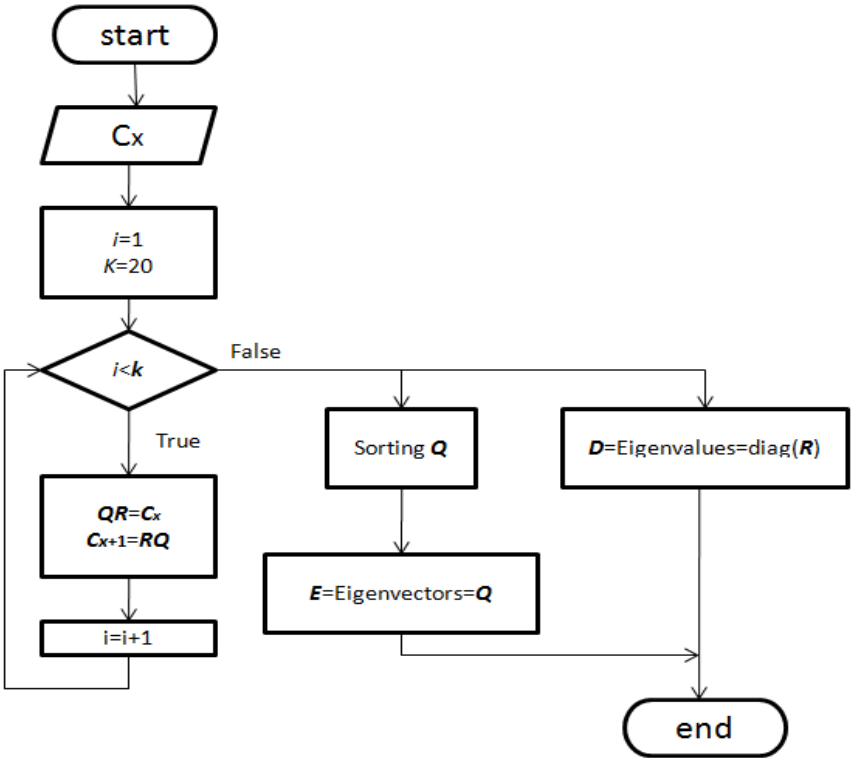


Figure 3.1: QR decomposition flowchart

According to Fig. 3.1, the eigenvalues are obtained by taking the diagonal elements of the $\boldsymbol{R}$ matrix. However, the eigenvectors require more steps to get the final $\boldsymbol{E}$ matrix.

The output of the flowchart given in Fig. 3.1 is two matrices $\boldsymbol{D}$ and $\boldsymbol{E}$ representing the eigenvalues and eigenvectors respectively. Their role is to obtain the whitening matrix $\boldsymbol{V}$ as given in Equation (3.4) given as follow:

$$\boldsymbol{V} = \boldsymbol{D}^{-1/2}\boldsymbol{E}^{T} = \begin{bmatrix} d_1^{-1/2} & 0 & 0 & 0 \\ 0 & d_2^{-1/2} & 0 & 0 \\ 0 & 0 & d_3^{-1/2} & 0 \\ 0 & 0 & 0 & d_4^{-1/2} \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & e_{13} & e_{14} \\ e_{21} & e_{22} & e_{23} & e_{24} \\ e_{31} & e_{32} & e_{33} & e_{34} \\ e_{41} & e_{42} & e_{43} & e_{44} \end{bmatrix}^{T} \tag{3.4}$$

In the equation, for each eigenvector of matrix $\boldsymbol{E}$ (e.g. $[e_{11}\ e_{12}\ e_{13}\ e_{14}]^T$) is represented by a column denoting the corresponding eigenvalue in matrix $\boldsymbol{D}$ ($[d_1^{-1/2} 0\ 0\ 0]^T$).

The uncorrelated output $\boldsymbol{Z}$ is obtained by multiplying the whitening matrix $\boldsymbol{V}$ obtained from Equation (3.4) by $\boldsymbol{X_{cen}}$. Fig. 3.2 shows the complete centering and whitening process using the QR decomposition.

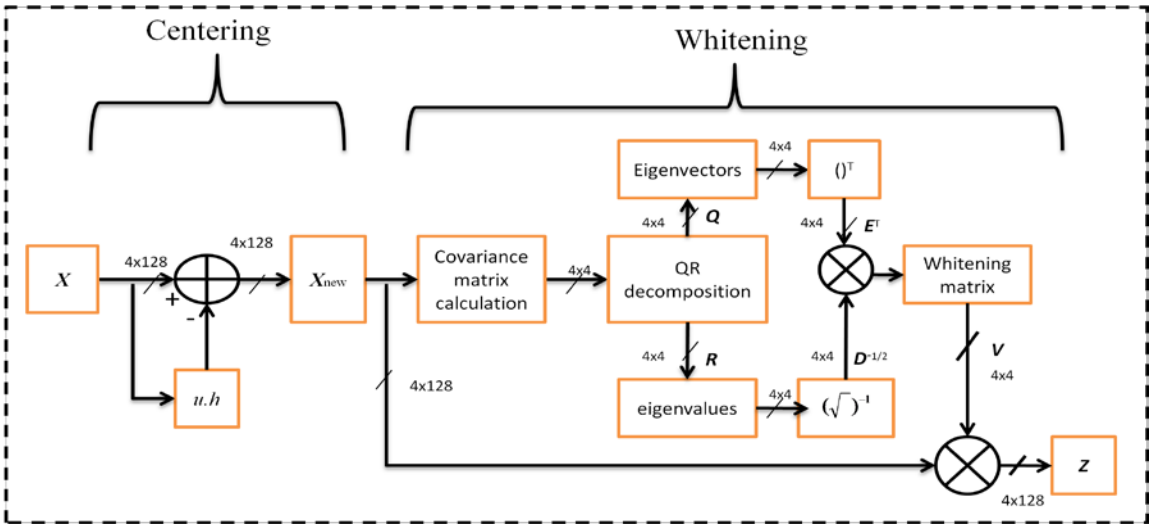


Figure 3.2: Whitening block diagram

### 3.4 FastICA using symmetric orthogonalization

This thesis focuses on implementing the FastICA algorithm by utilizing symmetric orthogonalization. This modification entails that the final unmixing matrix $\boldsymbol{W}$ (shown in Fig. 2.6 of Chapter 2) is $4 \times 4$ as given in Equation (3.1).

To begin with, finding the symmetric orthogonalization of the unmixing matrix $\boldsymbol{W}$ ($(\boldsymbol{W}\boldsymbol{W}^T)^{-1/2}\boldsymbol{W}$ given in Equation (2.20) requires calculating the term $(\boldsymbol{W}\boldsymbol{W}^T)^{-1/2}$. A standard algebraic method is to multiply the eigenvalues and eigenvectors $\boldsymbol{E}$ and $\boldsymbol{D}$ of the term $(\boldsymbol{W}\boldsymbol{W}^T)$ [31]. The problem is that for higher-order matrices (such as the $4 \times 4$ unmixing matrix $\boldsymbol{W}$ of this work), calculating the eigenvalues and eigenvectors $\boldsymbol{E}$ and $\boldsymbol{D}$ (both $4 \times 4$) is computationally-intensive [32]. What complicates the matter further is that this calculation must be performed iteratively in every step of the FastICA algorithm given in Chapter 2.

Alternative approaches have been proposed. One such approach is to use an iterative model to speed up the process of symmetrical orthogonalization. This approach was introduced in Chapter 2, Equations (2.22) and (2.23). Indeed, the iterative method converges to the same solution provided by the more expensive algerbraic method after less than 20 iterations as the simulation I performed given in Fig. 3.3 shows. The simulation given in the figure is separates the sources using the iterative method. The x-axis represents the number of iterations required for convergence while the y-axis represents the error between the current result of the iterative approach and the final result given by the algebraic approach. The simulation ends when the two solutions match giving zero error.

The figure does not mean that the iterative solution is slower than the algebraic one

because we are only modeling the number of iterations and not the time required to reach a solution. In fact, the 15 iterations performed by the iterative method converge to a solution much faster than the computations performed algebraically. This is a known property of iterative methods [34].

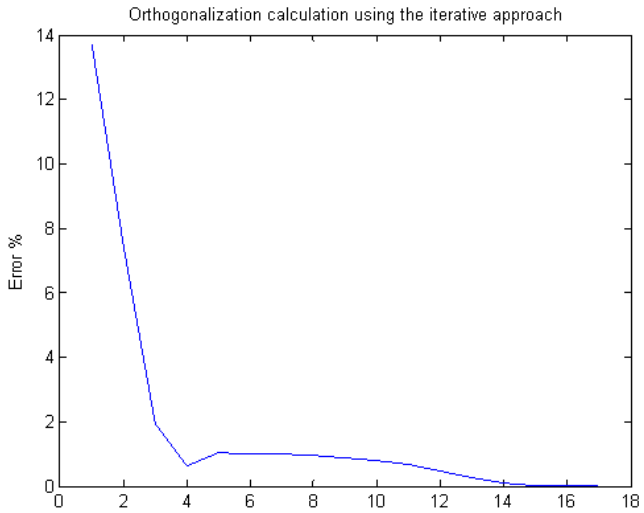


Figure 3.3: Symmetrical orthogonalization simulation using iterative approaches

## 3.5 FPGA implementation

It is well-known that the FPGA implementation of the FastICA algorithm is carried out using XILINX virtex5-XC5VLX50t FPGA chip. The LX50t chip has superior speed and larger area over the other virtex5 family [46]. The design is implemented using VHDL language. Since the system is designed to account for higher order data (four sensors), hierarchy is adopted throughout the design to provide a better control over the overall hardware structure and to monitor the overflow and underflow of each block.

Furthermore, implementation of DSP systems using floating-point arithmetic requires a huge hardware area and may lead to inefficient design especially for FPGA implementation [18]. On the other hand, fixed-point representation results in efficient hardware design. In this thesis, two's complement fixed-point arithmetic, is used. It

consists of an integer part and a fractional part as shown in Fig. 3.4.

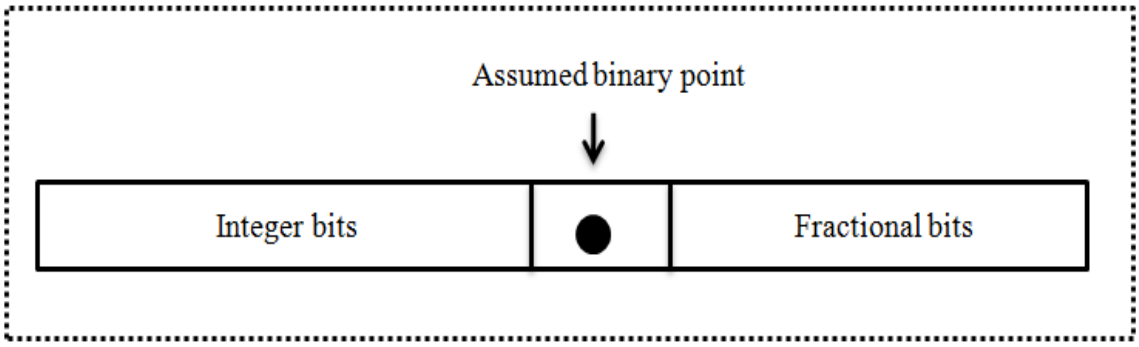


Figure 3.4: Fixed-point representation

The word length was selected based on several simulation attempts. Most of the results were faulty when a small word length was used since the small word length was not sufficient to represent the values. After several simulations attempts, the choice of the word length was decided not to be the same for various implementation blocks. For example, the QR decomposition block, the I/O and the intermediate signals word lengths were set to (26:13) which indicates 26 bits with 13 bits representing the integer part and 13 bits representing the fractional bits. This way, the integer part can represent numbers in the range of $2^{13} = 8192$. For the Centering and Covariance blocks, the word length was set to 16 bits because the calculation of the Centering and the Covariance were not complex and 16 bits were enough to represent for the intermediate variables like signals and storage elements within the implementation blocks. In general, the word lengths of the other blocks are listed in Table 3.1:

Table 3.1: Blocks word length used.

| BLOCK | Word length measured in bits | |
|---|---|---|
| | Integer | fractional |
| Centering | 8 | 8 |
| Covariance | 8 | 8 |
| QR decomposition | 13 | 13 |
| Whitening | 13 | 13 |
| Symmetric orthogonalization | 13 | 13 |
| One-unit FactICA | 13 | 13 |
| FastICA | 13 | 13 |

## 3.6 Hardware implementation

This Section describes the implementation stages of the complete system. The main block is divided into two stages namely Whitening and FastICA as shown in Fig. 3.5. Both stages have no control over each other. However, when the first stage, i.e. the whitening stage finishes its operation and the result is ready, the second stage is triggered by the main controller in Fig. 3.5. The main controller starts the process of the entire design; it enables and disables each block in the design based on the order of operation.

The controller consists of a finite state machine (FSM). The GO_FASTICA and GO_whitening signals are used to enable both stages in the design. In addition, CLK_whitening and CLK_FASTICA are the clocks supplied to Whitening and FastICA blocks. Address_sel_mem1 and CLK_mem1 are used for an intermediate RAM that holds the result of the Whitening stage and feed it to the FastICA block when required. Different clocks are used to reduce the power consumption and to provide a better control over the design. First, the controller activates the Whitening block to preprocess the signals and when the process is complete, the Whitening_busy signal becomes low allowing the controller to activate the FastICA. However, in order to pipeline the design, the Whitening block stays on after the process is complete to process another packet of data while the FastICA block is processing the first whitened packet of whitened data.

New_one control signal activates the whole process again when the FastICA finishes processing the first packet, the signal FastICA_Busy goes low when the first packet is processed by the FastICA to indicate that FastICA is ready to take another block of data from the Whitening block. Each packet contains $26 \times 128 \times 4$ bits of data stored in a ROM. Nevertheless, for the sake of simulation, only one packet of 128 samples is used in testing the implementation.

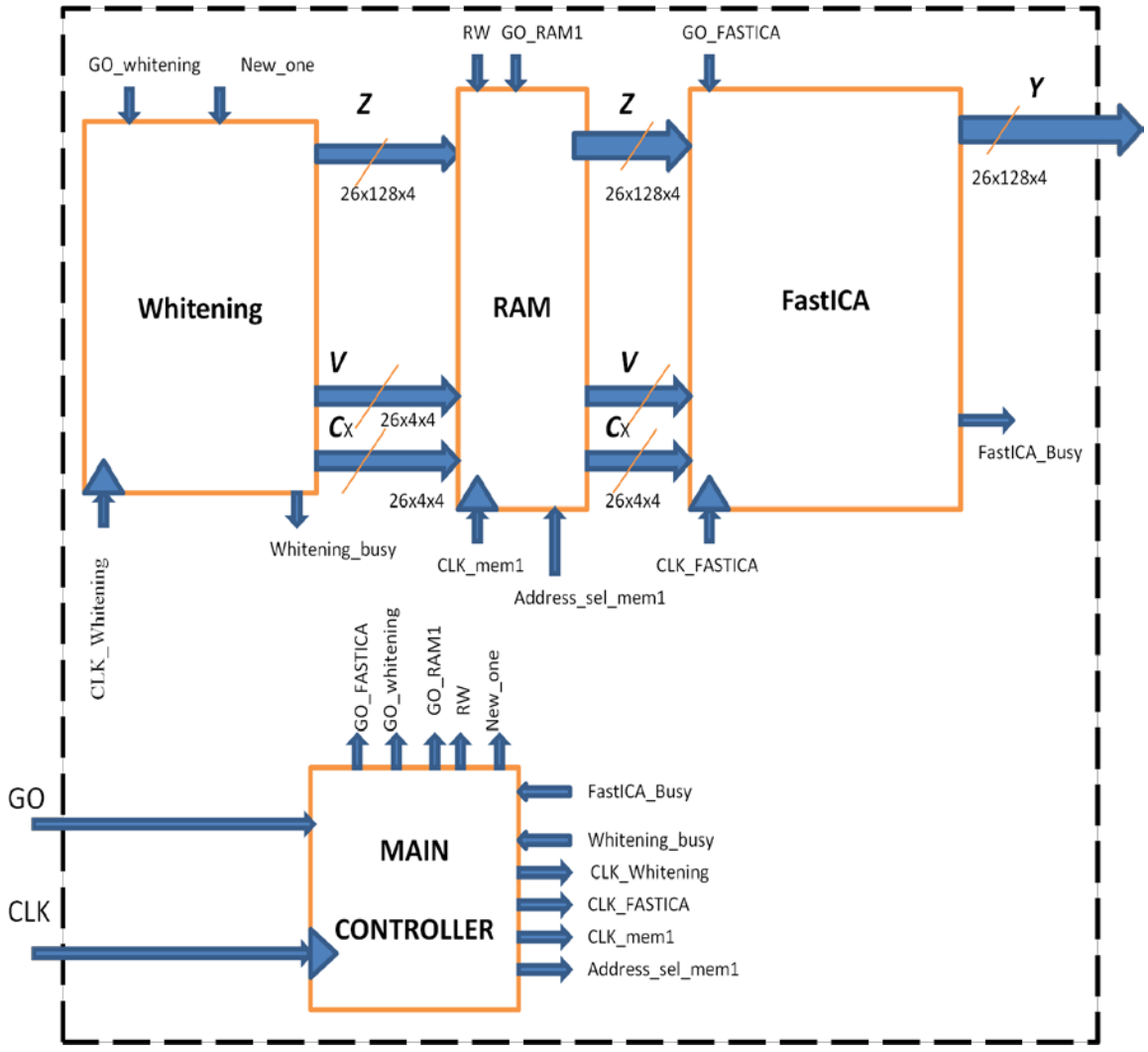


Figure 3.5: Main implementation block

Table 3.2: Complete system FPGA resources utilization report.

| Information | Count | Percentage Use |
|---|---|---|
| Slice Registers | 27779 of 28800 | 96% |
| Slice LUTs | 28403 of 28800 | 99% |
| Slice LUTs used as Logic | 28584 of 28800 | 99% |
| Slice LUTs used as RAM | 413 | |
| LUT Flip Flop pairs used | 27674 | |
| LUT Flip Flop pairs with an unused Flip Flop | 10689 of 27674 | 39% |
| LUT Flip Flop pairs with an unused LUT | 5432 of 27674 | 20% |
| Fully used LUT-FF pairs | 11553 of 27674 | 41% |
| Unique control sets | 326 | |
| IOs | 240 | |
| Bonded IOBs | 240 of 360 | 67% |
| BUFG/BUFGCTRLs | 26 of 32 | 81% |
| Block RAM/FIFO | 51 of 60 | 85% |
| DSP48Es | 45 of 48 | 94% |

Table 3.3: Complete system performance report.

| Clock name | Frequency response | MAX operating frequency | Estimated period | Input sampling |
|---|---|---|---|---|
| CLK | 20.0 MHz | 16.2 MHz | 62.5 ns | 1.857 KSPS |

Table 3.2 shows the complete system FPGA resources utilization report. In can be noticed that the system has been fully synthesized on a single FPGA chip since the area of implementation is still less than 28800 registers which is the available Slice registers in virtex5-XC5VLX50t. The total number of I/O pins used is 240 pins out of 360. Table 3.3 shows the maximum operating clock which is measured as 16.2 MHz when the system CLK is 20 MHz. The following sections describe the details of implementation of the algorithm.

### 3.6.1 Implementation of whitening

The Whitening block contains three stages namely, Centering, Covariance, and QR decomposition blocks as shown in Fig. 3.6.

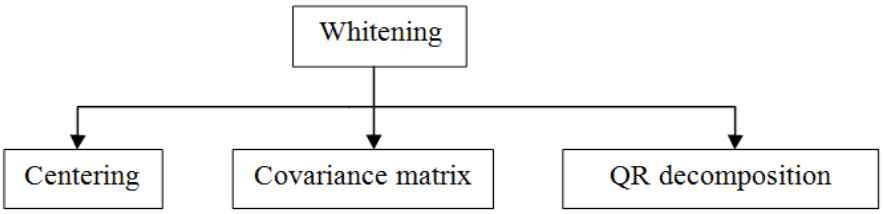


Figure 3.6: Whitening block

The first stage is the Centering block where the data is first fetched from the memory and the expected values are calculated and subtracted from the $\boldsymbol{X}$ according to Equation (2.2). The second stage calculates the covariance matrix of the centered signals while the third stage calculates the eigenvalues and eigenvectors of the covariance matrix. The MAIN CONTROLLER activates the Whitening block first. The Whitening implementation block is shown in Fig. 3.7. There are 128 samples fetched to the centering block for processing. RAM block 1 is activated to store $\boldsymbol{X}_{cen}$ after the data is centered. R_w1 and R_w2 are the controller's read and write operations in both RAM Modules. RAM Module 2 is in read mode as the Whitening result is available and the whiten_busy signal goes low. Multiplier 2 whitens the data by multiplying the $\boldsymbol{V}$ by $\boldsymbol{X}_{cen}$. RAM Module 2 keeps the whitened results for further analysis by other blocks. In addition, the data in RAM Module 2 will be available until another packet of information is processed and is ready to be written into RAM 2.

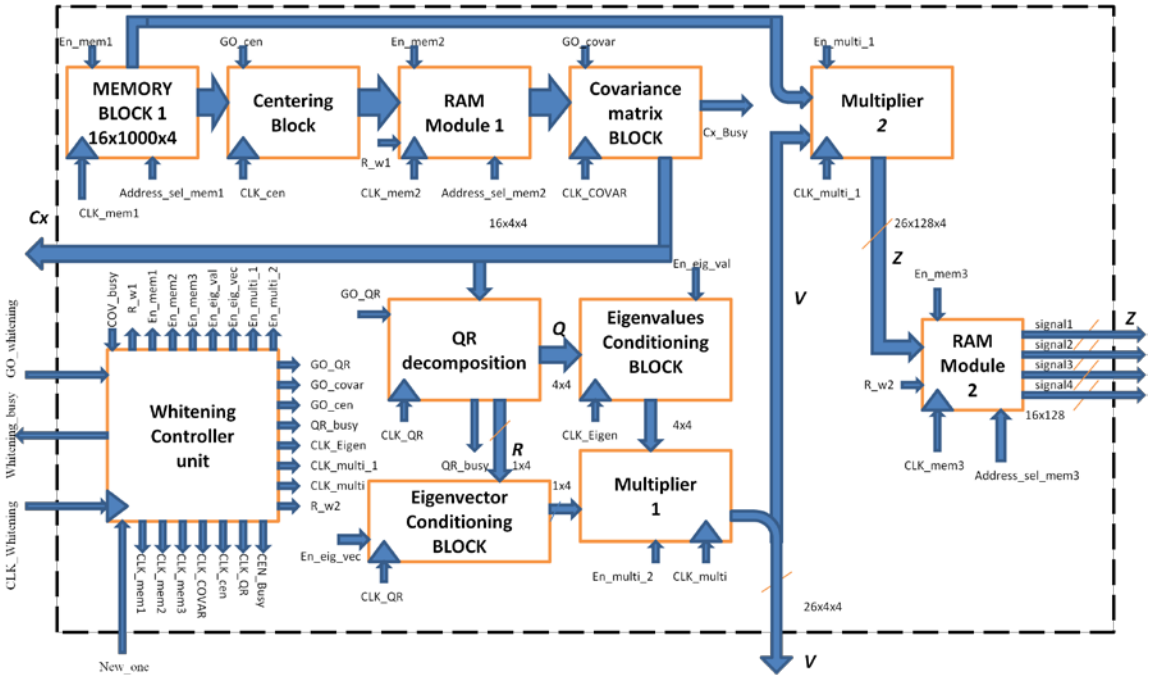


Figure 3.7: Whitening implementation block

Table 3.4 shows the resources utilization of the Whitening block when implemented separately. The overall area utilization is about 25% of the overall FPGA chip area. In addition, Table 3.5 indicates the Whitening maximum frequency, which is measured as 63 MHz when the block is simulated using the input CLK_Whitening set to 50 MHz . Table 3.6 is an extension of Table 3.5. It shows the timing details of CLK_Whitening through the Whitening clock.

Table 3.4: Whitening FPGA resources utilization report.

| Information | Count | Percentage Use |
|---|---|---|
| Slice Registers | 7407 of 28800 | 25% |
| Slice LUTs | 8382 of 28800 | 29% |
| Slice LUTs used as Logic | 8202 of 28800 | 28% |
| Slice LUTs used as RAM | 180 | |
| LUT Flip Flop pairs used | 10834 | |
| LUT Flip Flop pairs with an unused Flip Flop | 3427 of 10834 | 31% |
| LUT Flip Flop pairs with an unused LUT | 2452 of | 22% |

| | | |
|---|---|---|
| | 10834 | |
| Fully used LUT-FF pairs | 4955 of 10834 | 45% |
| Unique control sets | 368 | |
| IOs | 240 | |
| Bonded IOBs | 240 of 360 | 67% |
| BUFG/BUFGCTRLs | 1 of 32 | 3% |
| Block RAM/FIFO | 11 of 60 | 18% |
| DSP48Es | 27 of 48 | 56% |

Table 3.5: Whitening performance report.

| Clock name | Input frequency | MAX operating frequency | Estimated period | Input sampling |
|---|---|---|---|---|
| CLK_Whitening | 50.0 MHz | 62.189 MHz | 16.08 ns | 6.857 KSPS |

Table 3.6: Whitening timing report.

| Clock name | Path name | Estimated Frequency | Estimated period |
|---|---|---|---|
| CLK_Whitening | Input to Register | 135.2 MHz | 7.3964 ns |
| CLK_Whitening | Register to Register (worst case) | 62.189 MHz | 16.08 ns |
| CLK_Whitening | Register to Output | 265.4 MHz | 3.7679 ns |

It is worth noting that there is an apparent discrepancy between the input frequency in Tables 3.3 and 3.5. In order to explain this, I draw attention to the fact that for blocks varying in complexity, the maximum frequency that can be assigned to the block will vary. This is because each block can take a certain frequency after which the simulation will not be correct due to internal delays.

For example, in Table 3.3, the input frequency assigned to the complete system has to be low to account for all the blocks in the design to avoid timing problems. This is because if a higher frequency is used, the intermediate blocks will not produce the correct results in time for the following blocks to process the data. Using the same logic, a higher frequency was used in Table 3.5 as the system is simpler than that of Table 3.3 and will

therefore permit such increase. In the rest of this section, different input frequencies will be set to accommodate the design complexity accordingly.

### 3.6.1.1 Implementation of Centering block

The centering stage is the first sub block of the whitening operation. Centering means removing the mean of each input vector (128 samples) by subtracting the mean values from the original signals. The Centering stage contains 16-bit adders, 16-bit dividers and 16-bit subtractors. The mean of the four signals are calculated simultaneously since the signals are loaded at the same time to the Centering block as shown in Fig. 3.8. Table 3.7 also shows the mean gate-level results in comparison with the simulated MATLAB. According to Table 3.7, the results from the MATLAB simulation is considered very close to the gate-level simulation.

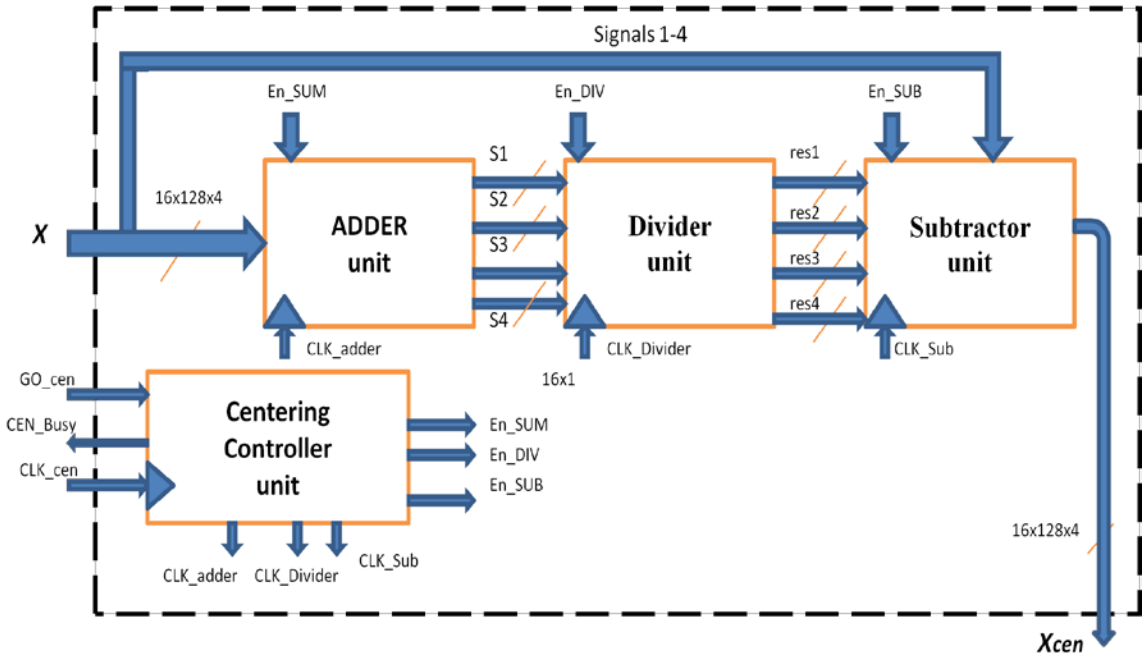


Figure 3.8: Implementation of centering

In addition, the accuracy of the Centering block can be increased by increasing the number of bits per word but since the block doesn't play a major rule in the

implementation process, 16 bits is considered adequate for all signals in the Centering block given in Fig. 3.8.

Table 3.7: Mean calculations result.

| Variable name | MATLAB Result | Gate-level Simulation |
|---|---|---|
| res1 | 0.3488 | 0.3486 |
| res2 | 0.2758 | 0.2756 |
| res3 | 0.3044 | 0.3041 |
| res4 | 0.2663 | 0.2661 |

The Centering Controller enables the three blocks in series using FSM. After the adder's result is available, a 16-bit divider is used to compute the mean of the four results over 128 samples according to Equation (2.4). Moreover, the ROM that holds the input requires 128 cycles to load the 128 input samples to the Centering block. In addition, the adder, the divider and the subtractor require 3 clock cycles to complete their task. According to the simulation results, the Centering output is available after 131 clock cycles. Table 3.8 shows the Centering FPGA resources utilization report. Table 3.9 shows the maximum frequency when the input the CLK_cen frequency is 100 MHz. It is measured as 150.7 MHz, in other words, this block cannot accept more than 150 MHz as CLK_cen. Table 3.10 provides more details about the CLK_cen path throughout the design.

Table 3.8: Centering FPGA resources utilization report.

| **Information** | **Count** | **Percentage Use** |
|---|---|---|
| Slice Registers | 2548 of 28800 | 8% |
| Slice LUTs | 1417 of 28800 | 4% |
| Slice LUTs used as Logic | 1417 of 28800 | 4% |
| LUT Flip Flop pairs with an unused Flip Flop | 369 of 2917 | 12% |
| LUT Flip Flop pairs with an unused LUT | 1500 of 10865 | 51% |
| Fully used LUT-FF pairs | 1048 of 10865 | 35% |
| Unique control sets | 146 | |
| IOs | 134 | |
| Bonded IOBs | 0 of 360 | 0% |
| Block RAM/FIFO | 4 of 60 | 6% |

Table 3.9: Centering performance report.

| Clock name | Input frequency | MAX operating frequency | Estimated period | Input sampling |
|---|---|---|---|---|
| CLK_cen | 100.0 MHz | 150.7 MHz | 6.6340 ns | 231.194 KSPS |

Table 3.10: Centering timing report.

| Clock name | Path name | Estimated Frequency | Estimated period |
|---|---|---|---|
| CLK_cen | Input to register | 538.2 MHz | 1.8580 ns |
| CLK_cen | Register to register(worst case) | 150.7 MHz | 6.6340 ns |
| CLK_cen | Register to output | 2123.1 MHz | 0.4710 ns |

### 3.6.1.2 Implementation of the covariance matrix

The covariance matrix in Equation (2.5) is realized in hardware using multiplier and divider units and a 16-bit register is used to hold the covariance result. Fig. 3.9 shows the Covariance matrix implementation. The Multiplier Module performs most of the calculations of the covariance matrix. According to Equation (3.3), 10 multiplications are required to calculate a $4 \times 4$ covariance matrix since there are 6 repeated elements in the covariance matrix. The Multipliers are based on the onboard XLINIX LogiCORE IP multiplier core [46]. XILINX multipliers reduce time and area in the FPGA chip resources. The XILINX IP multiplier takes 5 clock cycles to converge to the answer [46]. The multipliers have to process all the 128 samples, thus the multiplier module requires $128 \times 5$ clock cycles to converge to the answer. The adder/divider module uses the XILINX adder and the LogiCORE IP fixed-point divider v.4. The adder/divider module performs the $E[\cdot]$ operation in Equation (3.3). The COVAR Controller unit enables each block based on the availability of the result in every block.

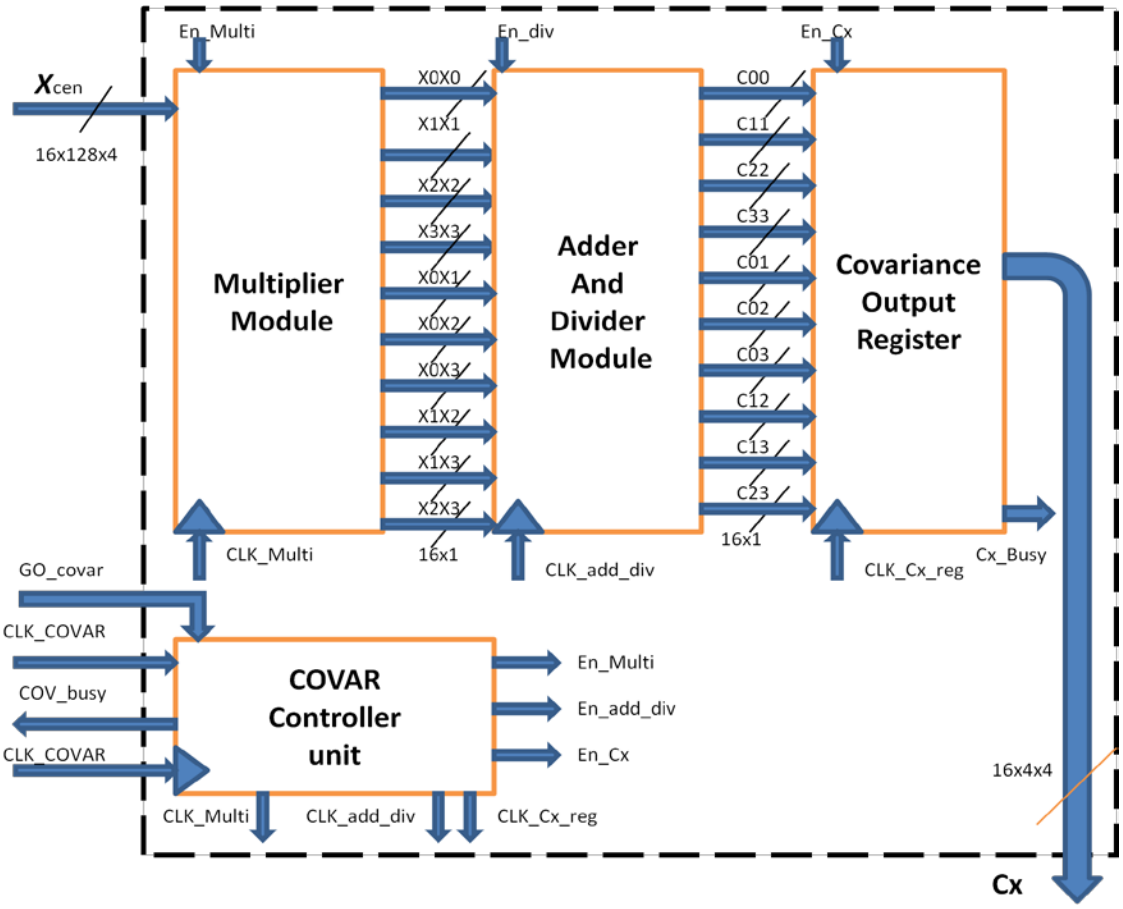


Figure 3.9: Implementation of the covariance matrix

The COV_busy goes low when the result is available by the covariance register. Table 3.11 shows the Covariance matrix implementation's simulation for the example in Fig. 2.3. The word length used is (16:7) bits where 7 bits are reserved for the fractional part. It can be seen that the result is correct up to three significant figures which is an acceptable result when used as input for the other blocks.

Table 3.11: Covariance matrix implementation result.

| $C_x$ | | | | MATLAB simulation | | | | Gate-level simulation | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $c_{11}$ | $c_{12}$ | $c_{13}$ | $c_{14}$ | 2.5563 | 1.2269 | 1.2417 | 1.2220 | 2.5557 | 1.2266 | 1.2412 | 1.2217 |
| $c_{21}$ | $c_{22}$ | $c_{23}$ | $c_{24}$ | 1.2269 | 1.2877 | 1.4621 | 1.2220 | 1.2266 | 1.2871 | 1.4614 | 1.2290 |
| $c_{31}$ | $c_{32}$ | $c_{33}$ | $c_{34}$ | 1.2417 | 1.4621 | 1.7307 | 1.3725 | 1.2414 | 1.4614 | 1.7300 | 1.3721 |
| $c_{41}$ | $c_{42}$ | $c_{43}$ | $c_{44}$ | 1.2220 | 1.2296 | 1.3725 | 1.1819 | 1.2217 | 1.2290 | 1.3721 | 1.1816 |

Table 3.12: Covariance matrix FPGA resources utilization report.

| Information | Count | Percentage Use |
|---|---|---|
| Slice Registers | 2228 of 28800 | 7% |
| Slice LUTs | 1165 of 28800 | 4% |
| Slice LUTs used as Logic | 1149 of 28800 | 3% |
| Slice LUTs used as Memory | 16 of 7680 | 0% |
| Slice LUTs used as RAM | 16 | |
| LUT Flip Flop pairs used | 2717 | |
| LUT Flip Flop pairs with an unused Flip Flop | 489 of 2717 | 18% |
| LUT Flip Flop pairs with an unused LUT | 1552 of 2717 | 57% |
| Fully used LUT-FF pairs | 676 of 2716 | 24% |
| Unique control sets | 149 | |
| Bonded IOBs | 149 of 360 | 41% |
| Block RAM/FIFO | 3 of 60 | 5% |
| DSP48Es | 1 of 48 | 2% |

Table 3.13 and 3.14 show the max frequency of the covariance matrix block when the input CLK_COVAR is 100 MHz.

Table 3.13: Covariance performance report.

| Clock name | Input frequency | MAX operating frequency | Estimated period | Input sampling |
|---|---|---|---|---|
| CLK_COVAR | 100.0 MHz | 158.3 MHz | 6.319 ns | 35.763 KSPS |

Table 3.14: Covariance timing report.

| Clock name | Path name | Estimated Frequency | Estimated period |
|---|---|---|---|
| CLK_COVAR | Input to register | 272.3 MHz | 3.6730 ns |
| CLK_COVAR | Register to register(worst case) | 158.3 MHz | 6.3190 ns |
| CLK_COVAR | Register to output | 2123.1 MHz | 0.4710 ns |

### 3.6.1.3 Implementation of QR decomposition

QR decomposition has two stages. The first stage is the implementation of the main QR decomposition and the second stage is reserved for rearranging the eigenvalues and eigenvectors. XILINX ACCELDSP tool 10.0 offers a complete pipelined QR decomposition block. Unfortunately, the block's results (***Q*** and ***R***) produced by

ACCELDSP tool are not aligned properly (each vector in $\boldsymbol{Q}$ does not correspond to the same vector in $\boldsymbol{R}$). Later this could cause the Whitening process to produce a faulty result (refer to equation 3.4). An extra stage is required to make sure that all the eigenvalues in $\boldsymbol{R}$ matrix correspond to all the eigenvectors in $\boldsymbol{Q}$. Fig. 3.7 shows the complete implementation of the Whitening block including the eigenvalues and eigenvectors conditioning blocks. The eigenvector conditioning block aligns and calculates $\boldsymbol{Q}^T$. The eigenvalue condition block aligns the $\boldsymbol{R}$ matrix and calculates the term $\boldsymbol{R}^{-1/2}$. It is important to mention that the matrix $\boldsymbol{R}$ has only diagonal elements and the rest of the elements in $\boldsymbol{R}$ are zeroes, which means that only 4 elements are needed from the $4 \times 4$ $\boldsymbol{R}$ matrix in the implementation. According to Equation (3.4) this reduces the calculation of the matrix $\boldsymbol{R}^{-1/2}$ to only calculate the elements $((r_{11})^{-1/2}, (r_{22})^{-1/2}, (r_{33})^{-1/2}, (r_{44})^{-1/2})$. Four XILIX LogiCORE parallel square root modules are used for this purpose. It is imperative to know that $\boldsymbol{Q}^{\mathrm{T}}$ and $\boldsymbol{R}^{-1/2}$ operations that are explained earlier are not part of the QR decomposition but they are whitening operations incorporated within the QR decomposition to pipeline the design, according to Equation (3.4). Tables 3.15-3.1 7 show the QR implementation and timing reports.

Table 3.15: QR FPGA resources utilization report.

| Information | Count | Percentage Use |
|---|---|---|
| Slice Registers | 1221 of 28800 | 4% |
| Slice LUTs | 2536 of 28800 | 8% |
| Slice LUTs used as Logic | 2516 of 28800 | 8% |
| Slice LUTs used as RAM | 20 | |
| LUT Flip Flop pairs used | 2808 | |
| LUT Flip Flop pairs with an unused Flip Flop | 1587 of 2808 | 56% |
| LUT Flip Flop pairs with an unused LUT | 272 of 2808 | 9% |
| Fully used LUT-FF pairs | 949 of 2808 | 33% |
| Unique control sets | 230 | |
| Bonded IOBs | 84 of 360 | 23% |
| BUFG/BUFGCTRLs | 1 of 32 | 3% |
| Block RAM/FIFO | 4 of 60 | 6% |
| DSP48Es | 18 of 48 | 37% |

Table 3.16: QR performance report.

| Clock name | Input frequency | MAX operating frequency | Estimated period | Input sampling |
|---|---|---|---|---|
| CLK_QR | 50.0 MHz | 76.5 MHz | 13.0730 ns | 10.657 KSPS |

Table 3.17: QR timing report.

| Clock name | Path name | Estimated Frequency | Estimated period |
|---|---|---|---|
| CLK_QR | Input to Register | 225.2 MHz | 4.4410 ns |
| CLK_QR | Register to Register (worst case) | 76.5 MHz | 13.0730 ns |
| CLK_QR | Register to Output | 306.4 MHz | 3.2640 ns |

The QR decomposition maximum frequency is 76.5 MHz when the input CLK_QR is set to 50 MHz. The maximum operating frequency of the Whitening block is dictated by the slowest block in the design. For example, the slowest block in the Whitening block is the QR decomposition. According to Table 3.5, the Whitening maximum frequency is 62 MHz when all the blocks are simulated together.

### 3.6.2 FastICA implementation

The FastICA block contains the Symmetric orth, the One-unit FastICA, NORM Divider and the error Calculation blocks as shown in Fig. 3.10. The Symmetric orth and the Norm Divider are the implementations of Equation (2.22) and (2.23) respectively.

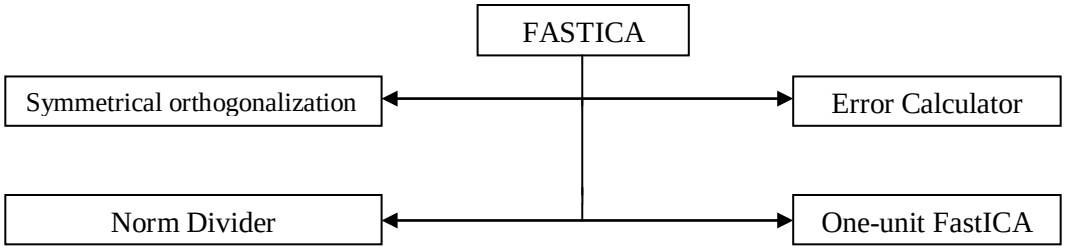


Figure 3.10: FastICA blocks

The Error Calculation block plays a major role in monitoring the convergence of the algorithm and decides wither to stop the separation or to keep the search active by

passing the result of the one-unit FastICA back to the Symmetric orth block. Once the term $\boldsymbol{W}^T\boldsymbol{W}$ is close to the identity matrix, the result is passed to multiplier Module 1 to get the separation matrix $\boldsymbol{W}$. However, if the result is not converged, the global stopping criterion is set to 70 iterations, which is in most cases more than enough for convergence. According to Equation (3.1), the separated signals are calculated by multiplying $\boldsymbol{W}$ by the whitened data $\boldsymbol{Z}$. This process is achieved by Multiplier Module 2. The Module contains 4 multipliers in parallel used to multiply the 128 whitened samples by the $4 \times 4$ separation matrix $\boldsymbol{W}$.

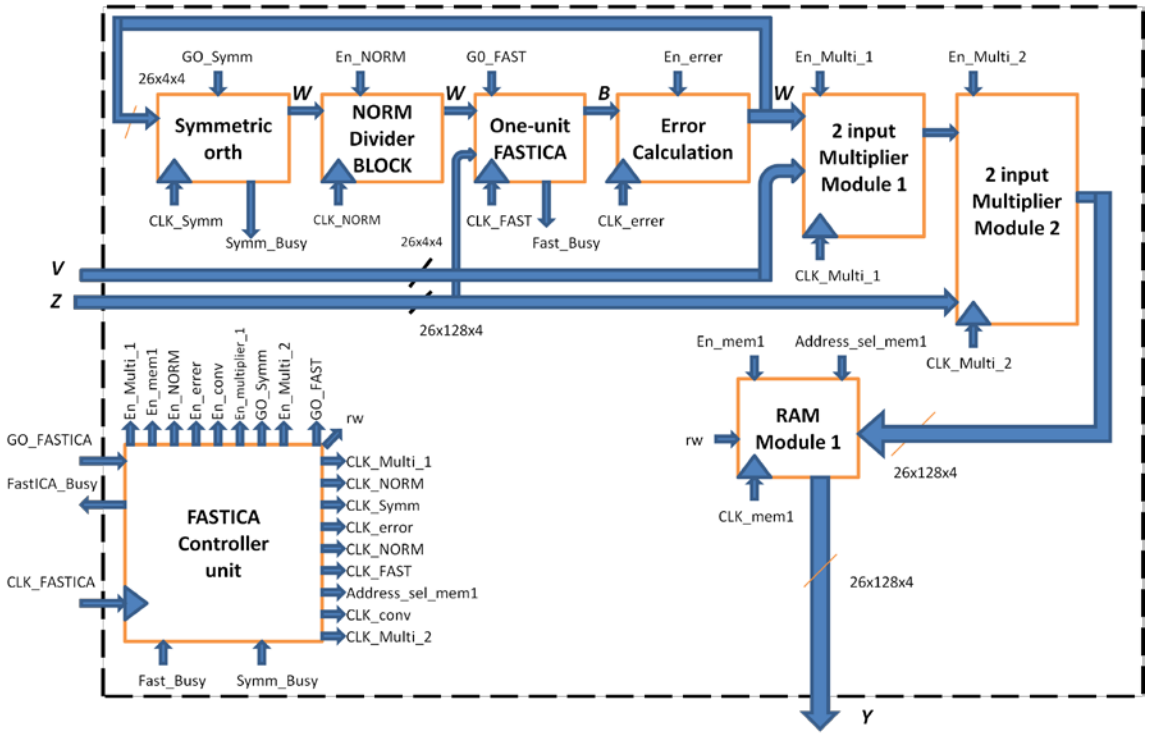


Figure 3.11: FastICA main block

The Norm Divider BLOCK consists of 26-bit fixed-point adder and divider that add the elements of $\boldsymbol{W}$ then divide each element of $\boldsymbol{W}$ by that value. The RAM block holds the result until another set of 128 samples is ready to be written, the rw signal controls the read and write operations of the RAM Module 1. The GO_FAST signal triggers the one-

unit FastICA algorithm to start the search using initial matrix $\boldsymbol{B_{initial}}$ . The matrix contains random arbitrary values that are stored in the One-unit FastICA. Table 3.18 shows the content of $\boldsymbol{B_{initial}}$.

Table 3.18: $\boldsymbol{B}$ initial condition.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $b_{11}$ | $b_{12}$ | $b_{13}$ | $b_{14}$ | -0.1493 | -0.5911 | 0.37934 | -0.1747 |
| $b_{21}$ | $b_{22}$ | $b_{23}$ | $b_{24}$ | 2.449 | -0.6547 | -0.3303 | -0.9573 |
| $b_{31}$ | $b_{32}$ | $b_{33}$ | $b_{34}$ | 0.473 | -1.0807 | -0.4999 | 1.2925 |
| $b_{41}$ | $b_{42}$ | $b_{43}$ | $b_{44}$ | 0.1169 | -0.0477 | -0.0359 | 0.4409 |

Once Fast_Busy signal becomes low, the error calculation block is activated to determine if the search termination condition is met. The symmetric orthogonalization is activated if the FastICA algorithm termination conditions are not met. The process is repeated until the algorithm is converged when the error equals to $|\boldsymbol{W}_+ - \boldsymbol{W}| < \varepsilon$ which is the difference between the current $\boldsymbol{W}_+$ and the previously calculated $\boldsymbol{W}$. The $\varepsilon$-value can be chosen to any value but for good approximation it is set to $0.001$ [33]. If $\varepsilon$ is not reached, the global maximum stopping criterion (GMSC) terminates the search. The GMSC was set to 70 iterations based on the results of several simulations. The GMSC role is to force the FastICA to quit searching if the algorithm cannot find an optimum $\boldsymbol{W}$ **(**refer to Fig. 2.5 for more details).

Table 3.19: FastICA FPGA resources utilization report.

| **Information** | **Count** | **Percentage Use** |
|---|---|---|
| Slice Registers | 19872 of 28800 | 69% |
| Slice LUTs | 20021 of 28800 | 70% |
| Slice LUTs used as Logic | 19982 of 28800 | 69% |
| Slice LUTs used as RAM | 240 | |
| LUT Flip Flop pairs used | 18544 | |
| LUT Flip Flop pairs with an unused Flip Flop | 3639 of 18544 | 19% |
| LUT Flip Flop pairs with an unused LUT | 4352 of 18544 | 24% |
| Fully used LUT-FF pairs | 10553 of 18544 | 57% |
| Unique control sets | 368 | |
| IOs | 240 | |

| | | |
|---|---|---|
| Bonded IOBs | 240 of 360 | 67% |
| BUFG/BUFGCTRLs | 17 of 32 | 53% |
| Block RAM/FIFO | 51 of 60 | 85% |
| DSP48Es | 27 of 48 | 56% |

Table 3.19 shows the resources utilization of the FastICA block. The overall FPGA chip area utilized is about 70%. In addition, Tables 3.20 and 3.21 show the maximum operating frequency which is 63 MHz when the input CLK_FASTICA is set to 20 MHz.

Table 3.20: FastICA performance report.

| Clock name | Input frequency | MAX operating frequency | Estimated period | Input sampling |
|---|---|---|---|---|
| Clock | 20.0 MHz | 23.3 MHz | 43.48 ns | 6.857 KSPS |

Table 3.21: FastICA timing report.

| Clock name | Path name | Estimated Frequency | Estimated period |
|---|---|---|---|
| Clock | Input to Register | 76.2 MHz | 0.135 ns |
| Clock | Register to Register (worst case) | 23.3 MHz | 43.448 ns |
| Clock | Register to Output | 165.4 MHz | 6.06 ns |

### 3.6.2.1 Implementation of One-unit FastICA

One-unit FastICA is the main building block of the FastICA algorithm and is the most computationally intensive block throughout the system. Besides the triggering inputs signals, the inputs are the whitened signals $\boldsymbol{Z}$ and result of the Symmetric orth block $\boldsymbol{W}$. Fig. 3.12 shows the implemented One-unit FastICA algorithm. Five multiplier modules, mean block and a subtractor module are used to implement the one-unit FastICA. B Decision Block is used to feed multiplier module 5 with the either $\boldsymbol{B_{initial}}$ or the result of the Symmetric orth block. Once the result is available by the subtractor unit, Fast_Busy signal goes low triggering the FastICA main controller to proceed and deactivate the unit. Deactivating the unit is very important in saving power since no clocks are fed to the one-unit FastICA, thus saving power.

Table 3.22: One-unit FastICA FPGA resources utilization report.

| Information | Count | Percentage Use |
|---|---|---|
| Slice Registers | 19272 of 28800 | 67% |
| Slice LUTs | 19821 of 28800 | 69% |
| Slice LUTs used as Logic | 19282 of 28800 | 67% |
| Slice LUTs used as RAM | 230 | |
| LUT Flip Flop pairs used | 18123 | |
| LUT Flip Flop pairs with an unused Flip Flop | 3439 of 18123 | 19% |
| LUT Flip Flop pairs with an unused LUT | 4152 of 18123 | 23% |
| Fully used LUT-FF pairs | 10353 of 18123 | 57% |
| Unique control sets | 368 | |
| IOs | 139 | |
| Bonded IOBs | 139 of 360 | 39% |
| BUFG/BUFGCTRLs | 17 of 32 | 53% |
| Block RAM/FIFO | 43 of 60 | 72% |
| DSP48Es | 20 of 48 | 42% |

Table 3.23: One-unit FastICA performance report.

| Clock name | Input frequency | MAX operating frequency | Estimated period | Input sampling |
|---|---|---|---|---|
| CLK_FAST | 20.0 MHz | 25.7 MHz | 40.06 ns | 8.857 KSPS |

Table 3.24: One-unit FastICA timing report.

| Clock name | Path name | Estimated Frequency | Estimated period |
|---|---|---|---|
| CLK_FAST | Input to Register | 78.2 MHz | 0.135 ns |
| CLK_FAST | Register to Register (worst case) | 25.7 MHz | 40.06 ns |
| CLK_FAST | Register to Output | 185.4 MHz | 5.4 ns |

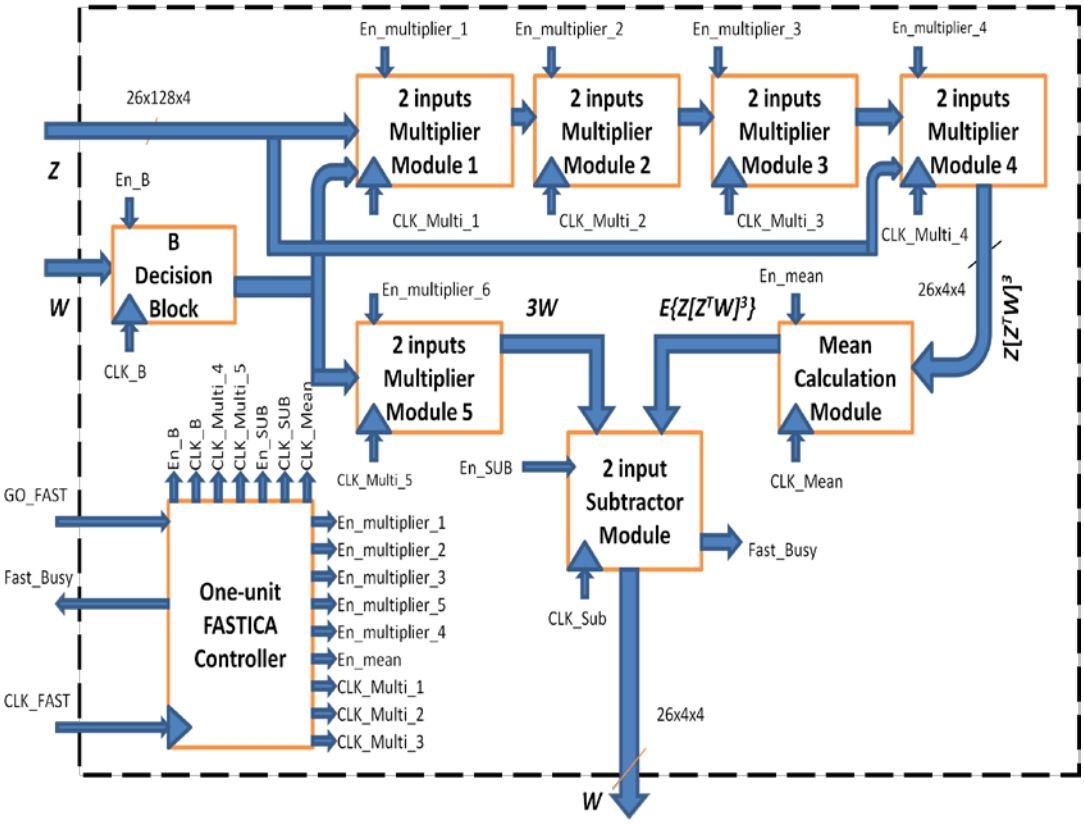


Figure 3.12: Hardware implementation of One-unit FastICA

### 3.6.2.2 Implementation of Symmetric orth

The implementation of Symmetric orth in Equations (2.22) and (2.23) are shown in Fig. 3.13. The design consists of 2 Multiplier Modules, NORM Calculator BLOCK, SQRT Calculator Module and Subtraction Module. The ***B*** matrix is fetched to the Symmetric orth block by the error calculation block in Fig. 3.11 to find a new orthogonal space. This is a required step in the FastICA algorithm so that the One-unit FastICA algorithm starts searching for another component in the mixing matrix ***W***. Otherwise, the One-unit FastICA will converge to the same weight matrix ***W***.

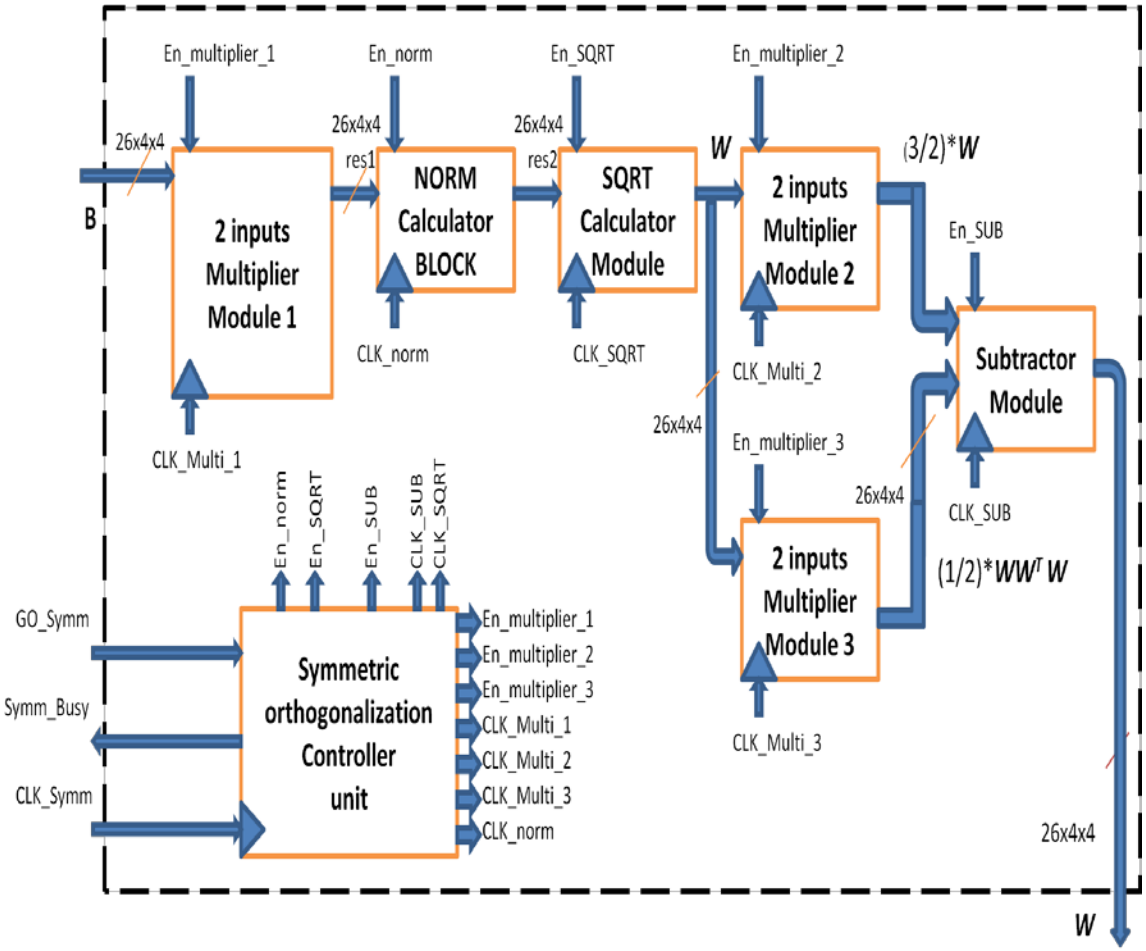


Figure 3.13: Implementation of Symmetric orth using iterative method

Table 3.25 shows the FPGA chip resources utilization report of the Symmetric orth block. The maximum operating frequency is almost 225 MHz when the input CLK_Symm is set to 100 MHz as in Table 3.26 while the timing report is provided in Table 3.27. It is observed that the estimated maximum frequency is about twice the input CLK_Symm, is due to the fact that numerical solution is used instead of the normally used matrix inversion in calculating the orthogonalization process (refer to chapter 2 for more details).

Table 3.25: Symmetric orth FPGA resources utilization report.

| Information | Count | Percentage Use |
|---|---|---|
| Slice Registers | 158 of 28800 | 0% |
| Slice LUTs | 199 of 28800 | 0% |
| Slice LUTs used as Logic | 199 of 28800 | 0% |
| LUT Flip Flop pairs used | 200 | |
| LUT Flip Flop pairs with an unused Flip Flop | 42 of 200 | 21% |
| LUT Flip Flop pairs with an unused LUT | 1 of 200 | 0% |
| Fully used LUT-FF pairs | 157 of 200 | 78% |
| Unique control sets | 41 | |
| IOs | 34 | |
| Bonded IOBs | 34 of 360 | 9.5% |
| BUFG/BUFGCTRLs | 1 of 32 | 3% |
| DSP48Es | 0 of 48 | 0% |

Table 3.26: Symmetric orth performance report.

| Clock name | Input frequency | MAX operating frequency | Estimated period | Input sampling |
|---|---|---|---|---|
| CLK_Symm | 100.0 MHz | 224.1 MHz | 4.4630 ns | 1120.032 KSPS |

Table 3.27: Symmetric orth timing report.

| Clock name | Path name | Estimated Frequency | Estimated period |
|---|---|---|---|
| CLK_Symm | Input to Register | 457.7 MHz | 2.1850 ns |
| CLK_Symm | Register to Register (worst case) | 224.1 MHz | 4.4630 ns |
| CLK_Symm | Register to Output | 2123.1 MHz | 0.4710 ns |

## 3.7 Summary

In this chapter, the proposed design and its implementation are presented. The proposed BSS model accepts up to 4 input sensors. It means that the model can separate up to four mixed signals in the mixture. The proposed Whitening and FastICA architectures and their FPGA implementations have been discussed. The proposed Whitening implementation is based on the QR decomposition. Also, the FastICA was designed based on an iterative symmetrical orthogonalization. The overall system was synthesized on a single virtex5-XC5VLX50t FPGA chip. The overall system operating frequency was measured as 16 MHz.

# Chapter 4

# *Simulation Results*

## 4.1 Introduction

This chapter is intended to show the Gate-level simulation of the proposed architecture. Since the algorithm is running in off-line mode (the input is stored on a ROM instead of obtaining them one by one by means of an analogue to digital convertor (ADC)). The results of the Gate-level simulations are compared with those obtained using MATLAB environment. Two experiments are conducted using the proposed architecture. The inputs are stored in memory BLOCK 1 in the Whitening block. The operating frequency used for both experiments is 10 MHz which is less than the maximum operating frequency, i.e.16 MHz.

## 4.2 Separating four signals

The first experiment involves the separation of four signals that are pre-mixed in MATLAB. The four signals have the following properties:

$$\text{Signal } 1 = randn(M, N) \tag{4.1}$$

$$\text{Signal } 2 = \text{square}(4\pi t) \tag{4.2}$$

$$\text{Signal } 3 = sin(2\pi t) \tag{4.3}$$

$$\text{Signal } 4 = sin(7\pi t) \tag{4.4}$$

where $t = [0\ ,0.1\ ,\dots,1.5]$, the signals are randomly mixed by adding them as follows:

$$X_1 = (\text{Signal 1} + \text{Signal 2}) \quad (4.5)$$

$$X_2 = (\text{Signal 3} + 0.6\ \text{Signal 2}) \quad (4.6)$$

$$X_3 = (\text{Signal 1} + 0.9\ \text{Signal 4}) \quad (4.7)$$

$$X_4 = (\text{Signal 2} + \text{Signal 4}) \quad (4.8)$$

Fig. 4.1 shows the signals before the mixing Equations. It is crucial to know the information of the signals beforehand so that the results can be compared with the actual signals after the algorithm is converged to the answer. Fig. 4.2 shows the signals after mixing them.

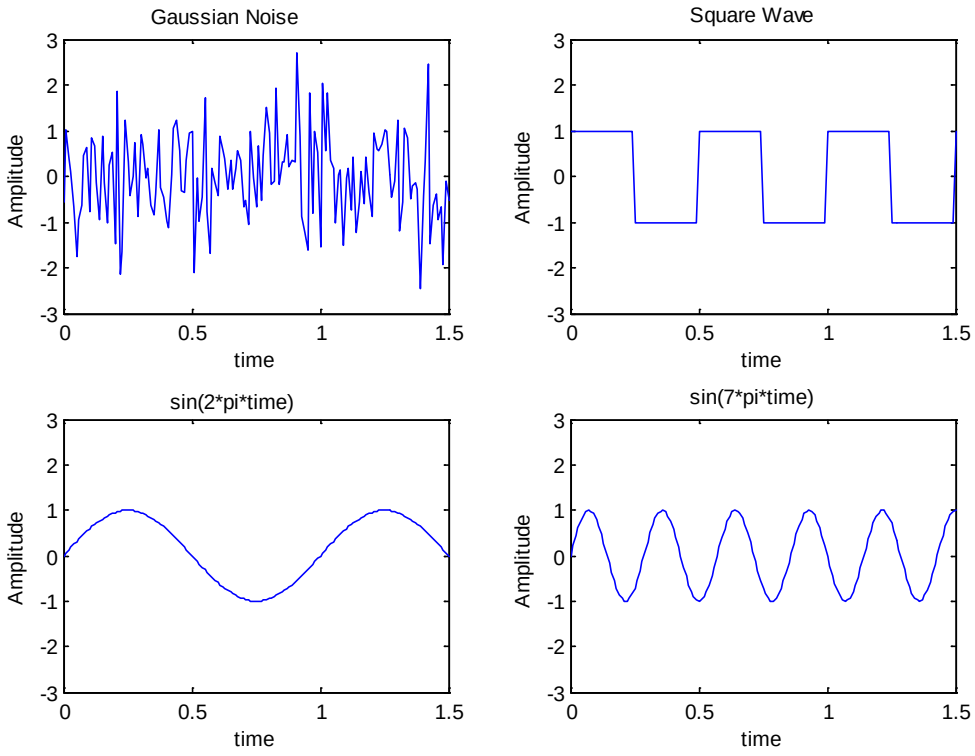


Figure 4.1: Four signals before mixing

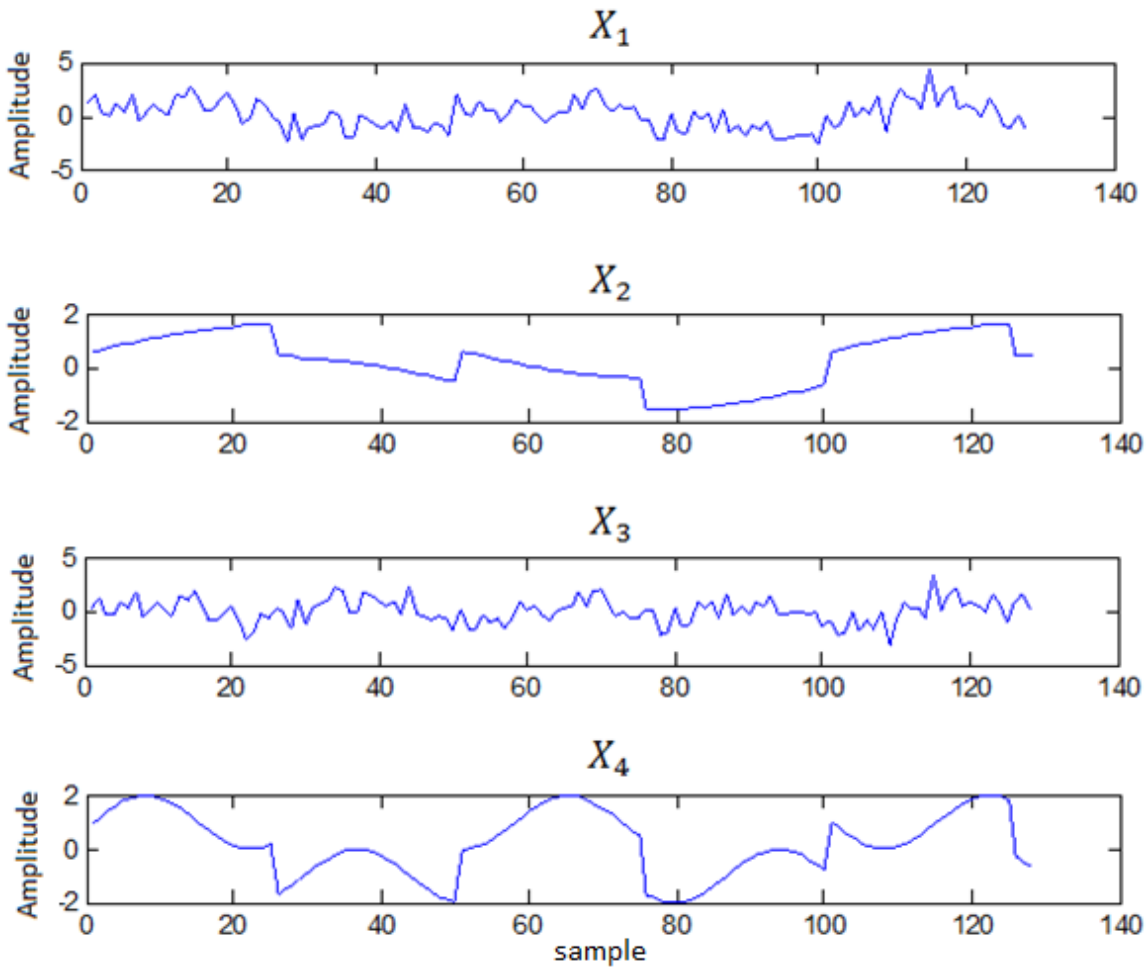


Figure 4.2: Four signals after mixing

Figures 4.3 and 4.4 show the results of the implementation of the algorithm to separate the four signals $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3$ and $\boldsymbol{X}_4$, each having a (26:13) word length. Fig 4.4 shows how whitening fails to separate the four signals. However, Figures 4.3 and 4.5 show the results of separation using the FastICA algorithm in both MATLAB and the gate-level simulations. It is clear that all four signals are separated. To measure the error between the MATLAB signals and the estimated signals, the square wave was examined since the error can be measured by subtracting the separated vectors $|\boldsymbol{y}_2| - |\boldsymbol{y}_4|$ that are shown in Figures 4.3(b) and Fig.4.5 (d), respectively. The difference between the two signals is less than 0.01 and is shown in Fig. 4.6.

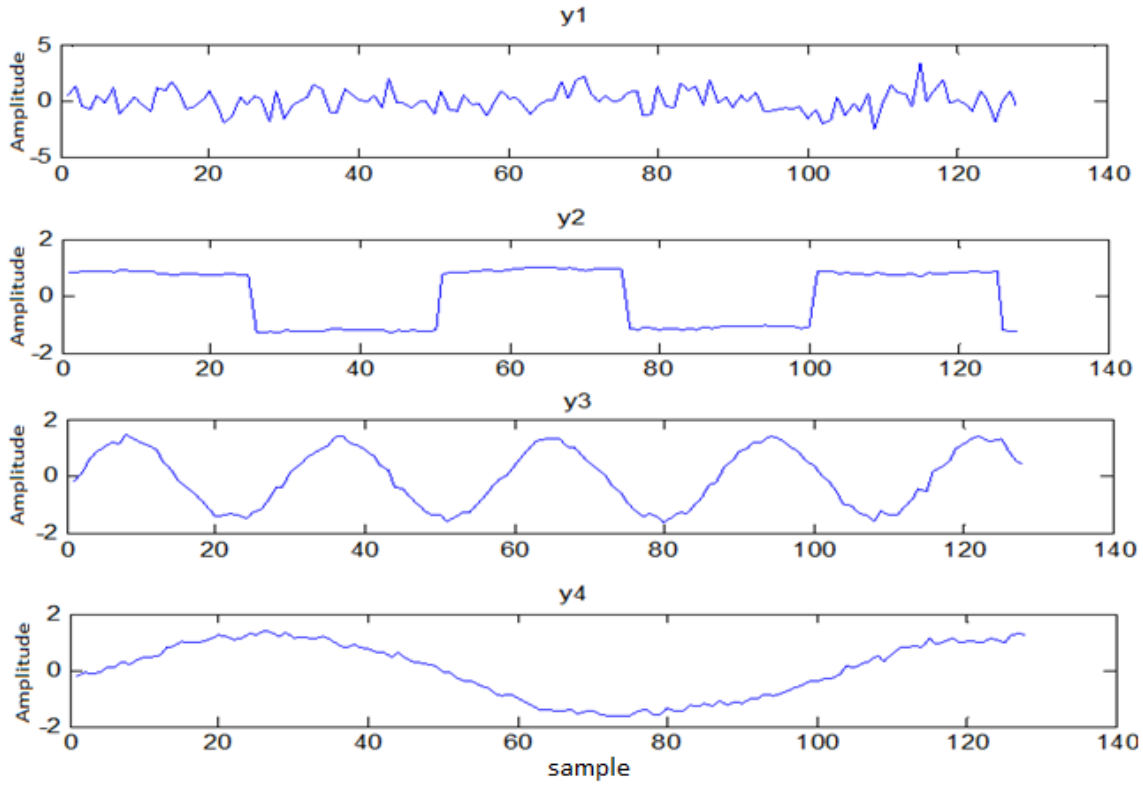


Figure 4.3: FastICA MATLAB simulation

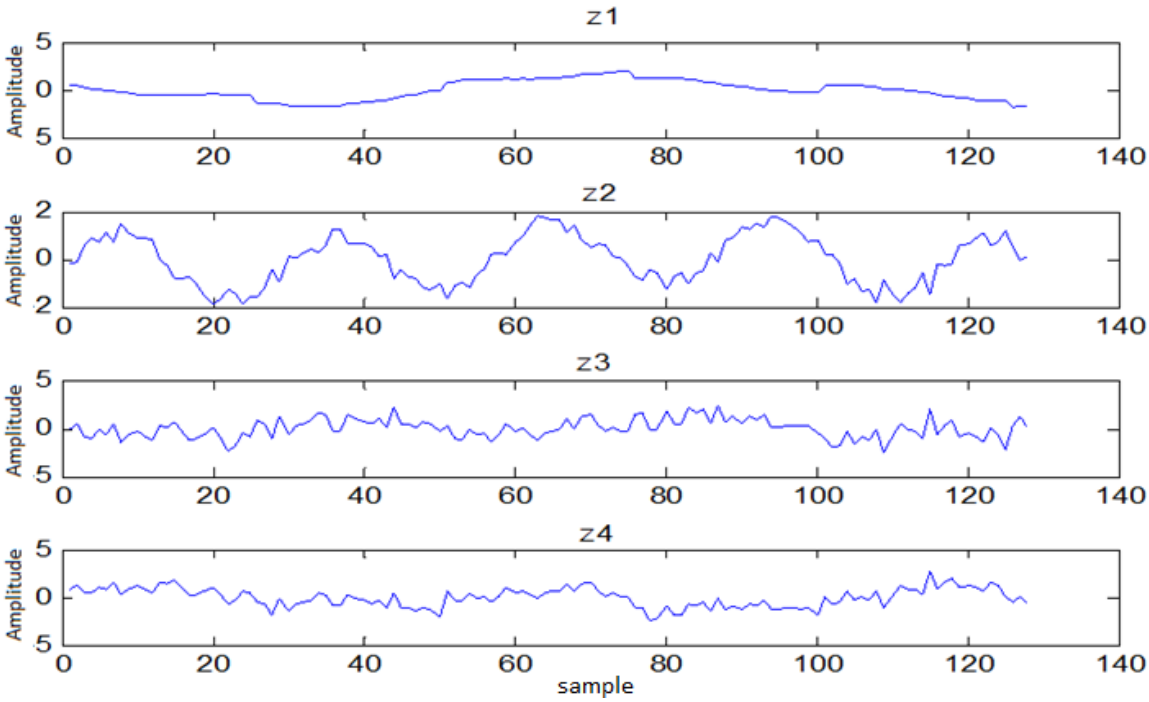


Figure 4.4: Whitening gate-level simulation

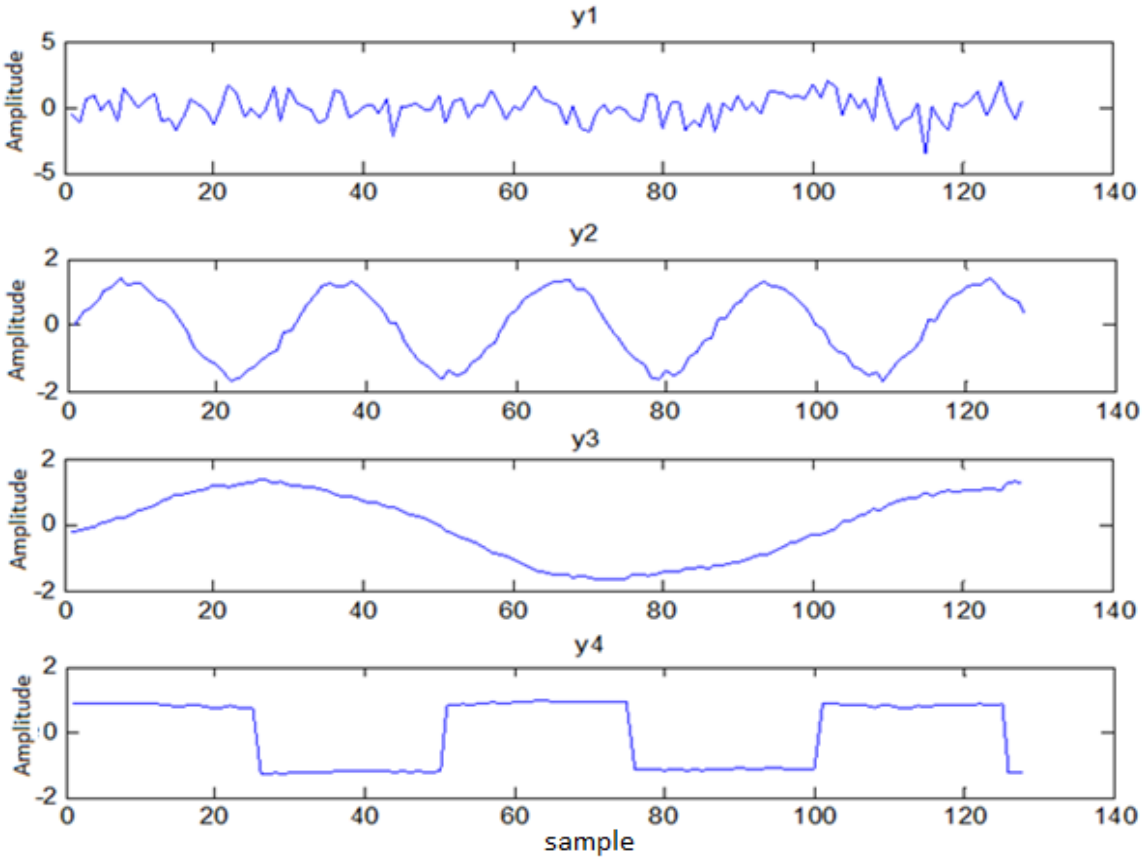


Figure 4.5: FastICA gate-level simulation

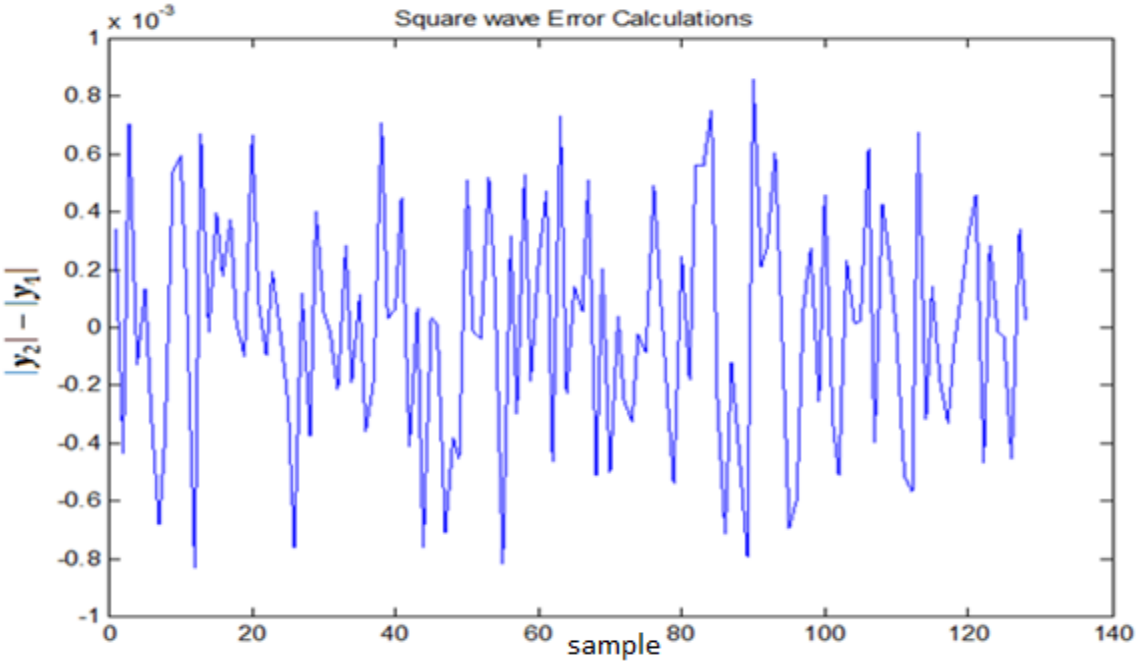


Figure 4.6: Square wave error analysis

### 4.3 Separating ECG signals

The main goal of this work is to separate four singles. Those signals are assumed to be taken from four sensors. In complex situations like the separation of the fetal ECG (FECG) from the mother ECG (MECG), more than two sensors are needed to better aid the separation process using FastICA algorithm [48-53]. Real ECG data taken from the American heart association [48] is used to test the performance of the proposed FastICA architecture. The data are taken from sensors placed on the abdominal and the thorax of a pregnant woman.

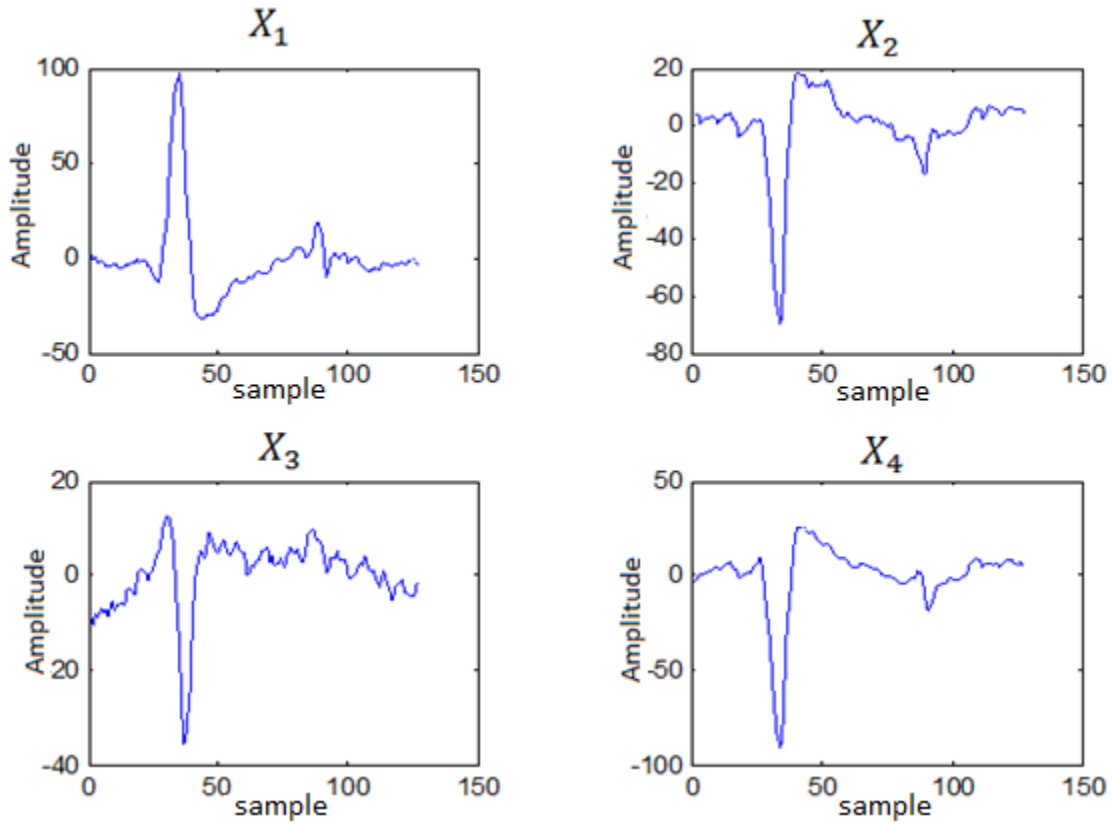


Figure 4.7: ECG signals [47]

The signals in Fig. 4.7 contain the MECG and FECG. It is difficult to separate the signals using only Whitening [53]. Also, it is difficult to compare the results of the separation with the predetermined or original signals like in the previous example [53, 55].

Table 4.1 shows the gate-level result of the final unmixing matrix $\boldsymbol{W}$. The simulations result is more or less close to the MATLAB simulation result. The error in the output is due to many factors like the quantization error, round off error since fixed-point representation is used, also overflow and underflow are very difficult to omit [20].

Table 4.1 Unmixing matrix result.

| **W MATLAB** | | | | **W Gate-level Simulation** | | | |
|---|---|---|---|---|---|---|---|
| -0.106 | -0.083 | 0.043 | -0.036 | -0.124 | -0.087 | 0.058 | -0.052 |
| -0.030 | -0.047 | 0.020 | -0.061 | -0.043 | -0.025 | 0.032 | -0.241 |
| 0.061 | -0.203 | 0.024 | 0.216 | 0.051 | -0.344 | 0.046 | 0.312 |
| -0.132 | -0.257 | -0.190 | 0.049 | -0.143 | -0.134 | -0.163 | 0.038 |

Fig. 4.8 and 4.9 show the result of multiplying the unmixing matrix $\boldsymbol{W}$ by $\boldsymbol{Z}$.

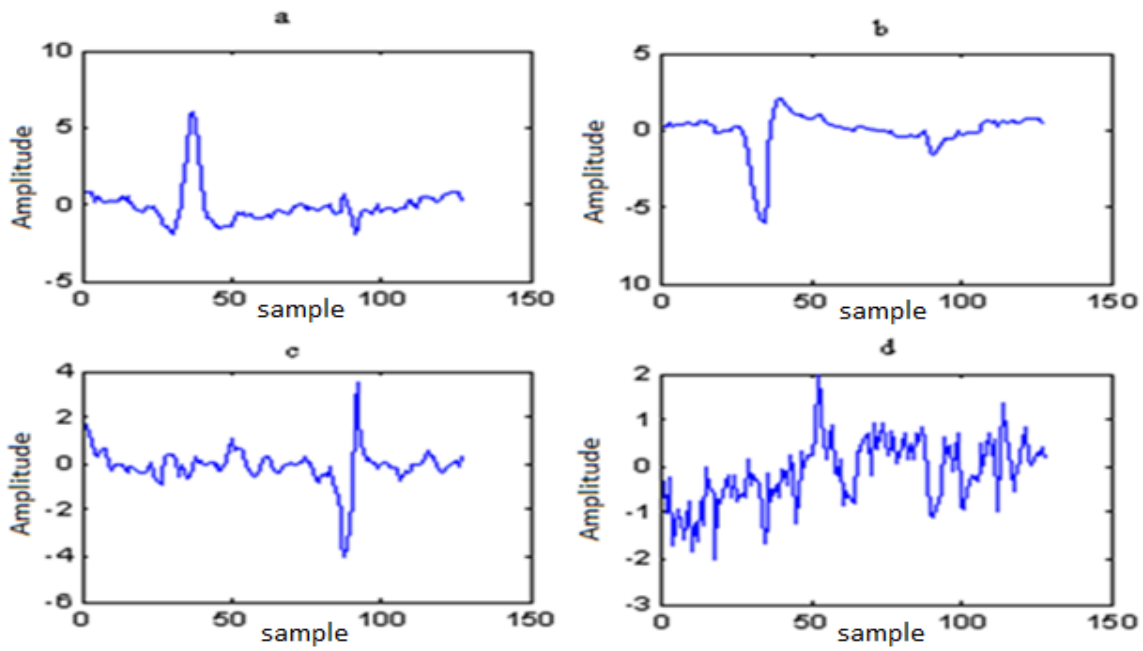


Figure 4.8: ECG separation simulation in MATLAB

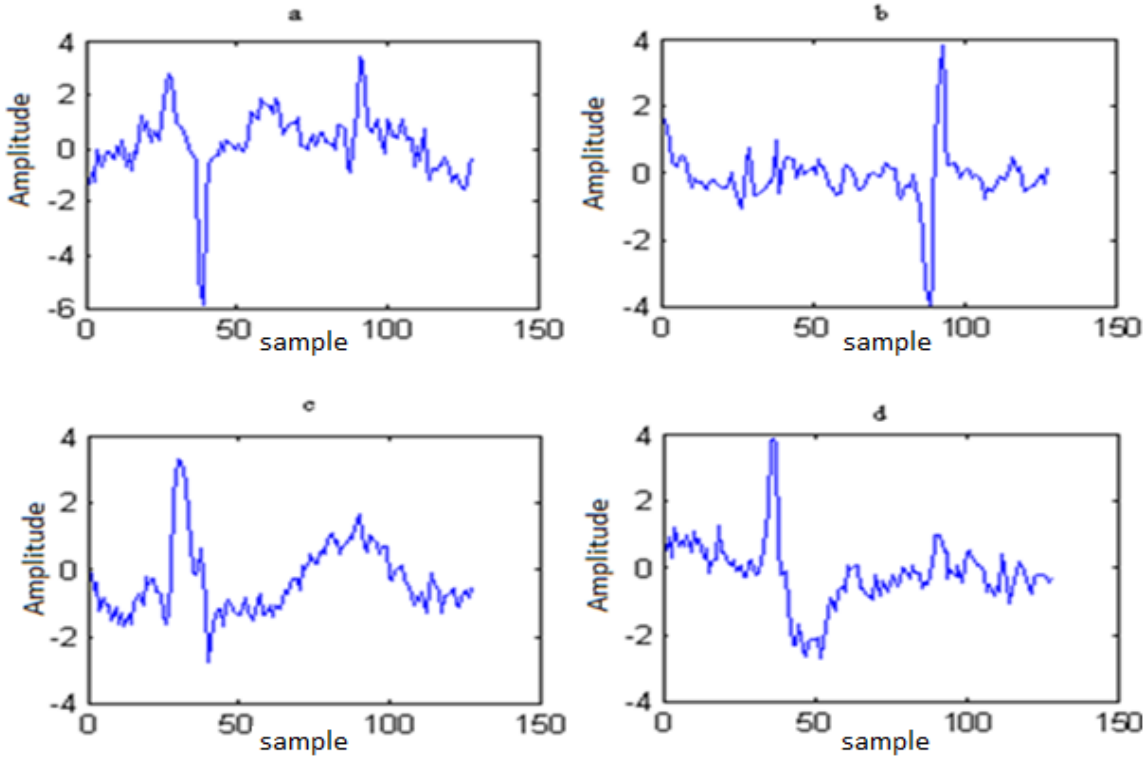


Figure 4.9: ECG gate-level simulation

According to [49, 51, 52], the main goal in FECG separation applications is to suppress the MECG and clarify the FECG so that doctors can diagnose the fetal heart health condition before birth [51]. The simulations given in Figures 4.8 and 4.9 show that the MATLAB and gate-level simulations did in produce one successful output out of four. For MATLAB, this successful output is given in Fig. 4.8(c), where one can clearly see FECG at almost the 70$^{th}$ sample. As for the other three MATLAB trials (Figures 4.8(a), 4.8(b) and 4.8(d)), this separation was not achieved. For instance, looking at Fig. 4.8(a) shows that the MATLAB output is very similar to the input signal given in Fig. 4.7(a). This is the case for the output given in Fig. 4.8(b), while the output given in Fig. 4.8(d) is simply jumbled signals.

For the gate-level simulation, the clarity of the FECG is shown in Fig. 4.9(b) while Figures 4.9(a), 4.9(c) and 4.9(d) the sharp appearance of FECG is not clear and the signals are not separated.

It is important to note that the separated signals given in Figures 4.8(c) and 4.9(b) are known to be in fact the FECG signals and not those of the mother because of the amplitude of the separated signal. The FECG signals has a much lower amplitude than that of MECG (the FECG amplitude is usually < 10 while the MECG amplitude is much higher than 10.

Another important point to note is the difference in the order of separation between MATPLAB and gate-level simulations. For example, MATLAB simulations show the FECG as the third separated signal while the gate-level simulation show the FECG as the second output. This is not an issue since the order of vectors $\boldsymbol{w_M}$ of the unmixing weight matrix $\boldsymbol{W}$ might change depending on the convergence of the FastICA. Figures 4.8(c) and 4.9(b) show the separated FECG, according to [48, 49], the FECG heart signal is considered clear and the mother ECG was completely removed.

Finally, there is an apparent difference in the shapes of the signals generated by MATLAB and gate-level simulations. A closer look at the figures however, shows that this difference does not exist when the algorithms succeed in separating the FECG as the successful simulations of Figures 4.8(c) and 4.9(b) have the same shape and no discrepancy between them exists. The rest of the figures did not succeed in separating the FECG and therefore have shapes that do not reflect the desired outcome.

Fig. 4.10 shows the absolute error analysis taken from the FECG in Fig. 4.8(c) and 4.9(b). The error in Fig. 4.10 is larger than the error in Fig. 4.6 due to the fact the ECG

signals are more complex than the previous example since ECG signals are not Gaussian in nature [52].

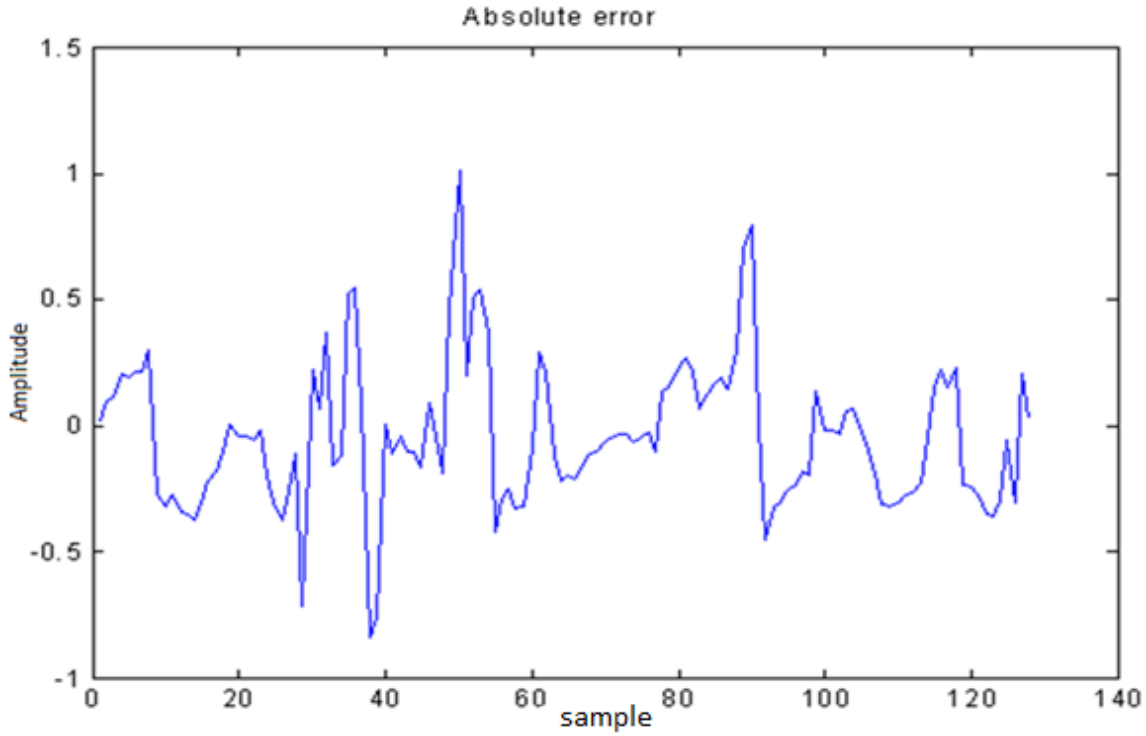


Figure 4.10: ECG absolute error analysis

## 4.4 Summary

This chapter describes the simulation results of FastICA algorithm. Four predetermined mixed signals are fed as input to the FastICA algorithm. The separation gate–level simulation results were compared to the MATLAB results and an error was calculated. The produced error is less than 0.01. The second simulation example was set to separate the FECG from the MECG signal. The separation was clear and the MECG was successfully suppressed. The MATLAB simulation and the gate-level simulations were compared to insure the functionality of the purposed FastICA implementation.

# Chapter 5

## *Conclusions and Future Work*

The goal of this thesis was to investigate the feasibility of implementing FastICA algorithm using four sensors. Increasing the number of sensors has a huge impact on the complexity of the algorithm and may render the hardware implementation impractical if the algebraic solution is used in both the preprocessing stage and in the main FastICA algorithm. To solve this issue, numerical solutions were used in the implementation of the system instead of the normally used algebraic method.

The implementation was carried out using the virtex 5 chip, which offers a variety of built-in fast multiplies, dividers and subtractors that were used intensively in this design. The system was fully implemented on a single chip. The maximum clock this system can use is 16 MHz. The ECG test signals were separated using four sensors readings. Real-time ECG separation can adopt this design since the design can be modified to account for real time applications. More than 128 samples can be added to the system after some modifications to the main controller. It is suffice to say that the proposed architecture can be extended account for more sensors to separate more complex applications.

The proposed architecture can be used as a building block to separate real-time signals by adding an analogue to digital converter before the Whitening block to acquire data and digital to analogue converter at the output to change the output back to analogue. The design can be further optimized by running the Whitening stage multiple times to process more samples before the FastICA busy signal goes low.

Given that the method presented here embodies a good solution for signal separation when four sources are mixed, the work can be extended by answering the question of how our method performs compared to algebraic methods when three or less signals are used. More specifically, we would like to find out ; 1) whether or not the performance gained using numerical methods is necessary when less signals are used 2) the conditions (if any) under which the numerical approach is more preferred compared to the algebraic solution.

Another important direction to follow is to extend the method presented here to account for online applications where the data is received and processed right away. Online applications require extra stages including ADC to process the input and a digital to analog convertor (DAC) to convert the output signals for display purposes. More memories are required to store the incoming packets from the ADC to be processed by the system.

Finally, this work can be used for signal separation in other applications such as Electroencephalography (EEG), electrical imaging of the heart, data mining and wireless communication applications.

# *References*


[1] B. Arons. A review of the cocktail party effect. Journal of the American Voice I/O Society, 12:35–50, July 1992.

[2] B. Arons. A review of the cocktail party effect. Journal of the American Voice I/O Society, 12:35–50, July 1992.

[3] J. V. Stone. Independent Component Analysis. MIT Press, 2004.

[4] P. Comon. Independent component analysis: A new concept? Signal Processing, 36:287–314, Apr. 1994.

[5] A. J. Bell and T.J. Sejnowski. An information maximization approach to blind separation and blind deconvolution. Neural Comput, 7:1129–1159, 1995.

[6] F. J. Theis. Mathematics in Independent Component Analysis. Lagos Verlag, 2002.

[7] J. T. Cobb. Statistical Properties of Synthetic Aperture Sonar Image Textures. In Proc. SPIE Conference, Orlando, Florida, March 17-20, 2008.

[8] N. Mitianoudis. Audio Source Separation using Independent Component Analy-sis. PhD thesis, Department of Electronic Engineering, Queen Mary, Universityof London, 2004.

[9] S. Amari, A. Cichocki, and H. H. Yang. A new learning algorithm for blind signal separation. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, Advances in Neural Information Processing Systems, volume 8, pages 757–763. The MIT Press, 1996.

[10] C. Cherry. Some experiments on the recognition of speech, with one and with two ears. Journal of Acoustical Society of America, 25:975–981, 1953.

[11] Ye (Geoffrey) Li,' Adaptive Blind Source Separation and Equalization for Multiple-Input/Multiple-Output Systems'. IEEE transaction on information theory, vol. 44, NO. 7, November 1998.

[12] Soliet, E.A., Gadallah, E. M. and Salah, A., Optimum detection of the fetus heart ECG signal, ICEENG l999.

[13] N.J.R. Muniraj and R.S.D. Wahidhabanu. A Novel Technique for Canceling Maternal ECG from Fetal ECG Using Virtex FPGA. Journal of Engineering and Applied Sciences 2 (5): 859-863, 2007.

[14] De Lathauwer L, Callaerts D, De Moor B, Vandewalle J. Fetal electrocardiogram extraction by source subspace separation. In proc. IEEE workshop on HOS. Girona, Spin, June 12-14 1995:134-138.

[15] C. M. Kim, H. M. Park, T. Kim, Y. K. Choi, and S. Y. Lee, "FPGA implementation of ICA algorithm for blind signal separation and adaptive noise canceling," IEEE Trans. Neural Netw., vol. 14, no. 5, pp.1038–1046, Sep. 2003.

[16] Hongtao Du and Hairong Qi "A Reconfigurable FPGA System for Parallel Independent Component Analysis". EURASIP Journal on Embedded Systems. volume 2006, Article ID 23025, Pages 1–12

[17] A. Celik, M. Stanacevic, and G. Cauwenberghs, "Mixed-signal realtime adaptive blind source separation," in Proc. IEEE Int. Symp. Circuits Syst., May 2004, vol. 5, pp. V-760–V-763.

[18] P. H. W. Leong and Leong Leong Cheung and O. Y. H. Cheung and T. Tung and C. M. Kwok and M. Y. Wong and K. H. Lee, "Pilchard -- A Reconfigurable Computing Platform with Memory Slot Interface", in Proc IEEE Symp. Field-Programmable Custom Computing Machine (FCCM), 2001, pp. 170-179

[19] Kuo-kai "FPGA Implementation of fastICA based on Floating-point Arithmetic design for real- Time Blind source Separation". Neural Networks, 2006. IJCNN '06. International Joint Conference on, 2785 – 2792.2006.

[20] Kevin Banovi´c , "Blind Adaptive Equalization for QAM Signals: New Algorithms and FPGA Implementation", University of Windsor, Master's Thesis, 2006.

[21] U. Meyer-Baese, Digital Signal Processing with Field Programmable Gate Arrays,2nd ed., ser. Signals and Communication Technology Series. New York, USA: Springer-Verlag, February 2007.

[22] D. Pellerin and S. Thibault, Practical FPGA programming in C. Englewood Cliffs, N.J., USA: Prentice-Hall, 2005.

[23] Aapo hayvarinen "Independent Component analysis". Adaptive and Learning Systems for Signal Processing, Communications, and Control Published Online: 15 May 2002.

[24] J.-F. Cardoso. Blind signal separation: Statistical principles. Proceedings of the IEEE, 86:2009–2025, Oct. 1998.

[25] Hotelling, H., Analysis of complex of statistical variables into principal components. Journal of educational psychology, 1993.24:p.417-441,498-520.

[26] A. L. Garcia. Probability and Random Processes for Electrical Engineering. Addison-Wesley, 2nd Edition, 1994.

[27] W. E. Arnoldi, The principle of minimized iteration in the solution of the matrix eigenvalue problem, Quart. Appl. Math. 9 (1951), 1729.

[28] I. T. Jolliffe. Principal Component Analysis. Springer-Verlag, NY, 2002.

[29] Xiaojun wang, Miriam leeser. A Truly Two-Dimensional Systolic Array FPGA Implementation of QR Decomposition. ACM Transactions on Embedded Computing Systems (TECS). Volume 9, Issue 1 October 2009.

[30] Y. Saad, Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices, Lin. Alg. App. 34 (1980), 269295.

[31] A. Papoulis. Probability, random variables and stochastic processes. McGraw-Hill, 2 edition, 1984

[32] A. Hyv¨arinen. One-unit learning rules for independent component analysis: A statistical analysis. Advances in Neural Information Processing Systems, 9:480–486, 1997.

[33] A. Hyv¨arinen, J. Karhunen, and E. Oja. Independent Component Analysis. John Wiley Sons, 2001.

[34] A. Hyv¨arinen. Fast and robust fixed-point algorithms for independent component analysis. IEEE Trans. Neural Networks, 10:626–634, May 1999.

[35] C. Jutten and J. Herault. Blind separation of sources, part i: An adaptive algorithm based on neuromimetic architecture. Signal Processing, 24:1–10, 1991

[36] S. Haykin. Neural Networks - A Comprehensive Foundation. Prentice Hall, 2nd edition,1998.

[37] A. Cichocki and S. Amari. Adaptive Blind Signal and Image Processing,. Wiley, 2002.

[38] C. M. Bishop. Neural Networks for Pattern Recognition. Clarendon Press, 1995.

[39] A. Hyv¨arinen. Beyond independent components. In Proc. Int. Conf on Articial Neural Networks, Edinburgh, UK, pages 809–814, 1999.

[40] A. Hyv¨arinen, P. Hoyer, and E. Oja. Sparse code shrinkage: denoising by nonlinear maximum likelihood estimation. In Proceedings of the 1998 conference on Advances in neural information processing systems II, pages 473–479, Cambridge, MA, 1999. MIT Press.

[41] S. C. Douglas. Blind source separation and independent component analysis: A crossroads of tools and ideas. In 4th International Symposium on Independent component analysis and blind signal separation (ICA2003), pages 1–10, Nara, Japan, 2003.

[42] A. Hyv¨arinen. Survey on independent component analysis. Neural Computing Surveys, 2:94–128, 1999.

[43] A. Hyv¨arinen. A family of fixed-point algorithms for independent component analysis. In Proceedings IEEE Int. Conf. on Acoustic, Speech, and Signal Processing (ICASSP'97), pages 3917–3920, Munich, Germany, 1997.

[44] S. C. Douglas. Fixed-point fastica algorithms for the blind separation of complex valued signal mixtures. In Signals, Systems and Computers, 2005. Conference Record of the Thirty-Ninth Asilomar Conference, pages 1320– 1325, Pacific Grove, CA, 2005.

[45] H. Hu, "Positive definite constrained least-squares estimation of matrices," Linear Algebra and it's Applications, vol. 229, pp. 167–174, 1995.

[46] H. Hu and I Olkin, "A numerical procedure for finding the positive definite matrix closest to a patterned matrix," Statistical and Probability Letters, vol. 12, pp. 511–515, 1991.

[47] Xilinx datasheets 2010 , accessed 19 Jan 2010. http://www.xilinx.com/onlinestore/silicon/online_store_v5.htm

[48] Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PCh, Mark RG, Mietus JE, Moody GB, Peng CK, Stanley HE. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. Circulation 101(23):e215-e220, 2000.

[49] McSharry P E, Clifford G D, Tarassenko L and Smith L A 2003 A dynamical model for generating synthetic electrocardiogram signals IEEE Trans. Biomed. Eng. 50 289–94

[50] Vanderschoot J, Callaerts D, Sansen W, Vandewalle J, Vantrappen G and Janssens J 1987a Two methods for optimal MECG elimination and FECG detection from skin electrode signals IEEE Trans. Biomed. Eng. 34 233–43

[51] Vanderschoot J, Callaerts D, Sansen W, Vandewalle J, Vantrappen G and Janssens J 1987b Two methods for optimal MECG elimination and FECG detection from skin electrode signals IEEE Trans. Biomed. Eng. 34 233–43

[52] Westgate J A, Bennet L and Gunn A 2002 The role of fetal ECG monitoring in labour Fetal Matern. Med. Rev. 13 119–39

[53] Zarzoso V, Nandi A and Bacharakis E 1997 Maternal and foetal ECG separation using blind source separation

[54] Kanjilal, P. P., Palit, S. And Saha, G., Fetal ECG Extraction using singular value decomposition, IEEE, Vol. BME-44, N0.1, January 1997 methods IMA J. Math. Appl. Med. Biol. 14 207–25

[55] Francisco Castells, Pablo Laguna," Principal Component Analysis in ECG SignalProcessing"Hindawi Publishing Corporation EURASIP Journal on Advances in Signal ProcessingVolume 2007, Article ID 74580, 21 pages

# *Appendix A*

Considering $\boldsymbol{W}_+$ to be the result of applying once the iteration step in 2.21 on $\boldsymbol{W}$. Let $\boldsymbol{WW^T = EDE^T}$ be the eigenvalue decomposition of $\boldsymbol{WW}^{\mathrm{T}}$. Then we have

$$\boldsymbol{W}_+\boldsymbol{W^T} = \frac{9}{4}EDE^T - \frac{3}{2}ED^2E^T + \frac{1}{4}ED^3E^T \quad \text{(A.1)}$$

$$= E(\frac{9}{4}D - \frac{3}{2}D^2 + \frac{1}{4}D^3)E^T \quad \text{(A.2)}$$

It is imperative to know that due to normalization in 2.22, all the eigenvalues of $\boldsymbol{WW}^{\mathrm{T}}$ are in the interval [0,1]. 2.23 shows that for every eigenvalue of $\boldsymbol{WW}^{\mathrm{T}}$, say $\lambda_i$, $\boldsymbol{W}_+\boldsymbol{W}_+^{\boldsymbol{T}}$ has a corresponding eigenvalue $h(\lambda_i)$ where $h(\cdot)$ is defined as:

$$h(\lambda) = \frac{9}{4}\lambda - \frac{3}{2}\lambda^2 + \frac{1}{4}\lambda^3 \quad \text{(A.3)}$$

Therefore, after $k$ iterations, the eigenvalues of $\boldsymbol{WW}^{\mathrm{T}}$ are obtained as $h(h(h(\ldots h(\lambda_i)))$, where $h$ is applied $k$ times on the $\lambda_i$, which are the eigenvalues of $\boldsymbol{WW}^{\mathrm{T}}$ for the original matrix before the iterations.

# *VITA AUCTORIS*

AL-laith taha was born in Baghdad, Iraq on November 5, 1982. He received his B.Eng. degree in electrical and electronics engineering in 2006 from the University of Northumbria at Newcastle, UK. He received his M.Eng in electrical and computer engineering from the University of Windsor in 2008. He is currently a candidate in the electrical and computer engineering M.A.Sc. program at the University of Windsor. His research interests include blind source separation for non-Gaussian signals, adaptive signals processing, FPGA implementation of DSP algorithms. Computer arithmetic and high-performance VLSI circuit design.